INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Escuela Superior de Ciencias Experimentales y Tecnología

Curso académico 2006-2007

Proyecto Fin de Carrera

Reconstrucción 3D visual con triangulación

Autor: José María Esteban Pacios

Tutor: José María Cañas Plaza

En este mundo sólo hay dos cosas infinitas: el universo y la estupidez humana. y del universo aún no estamos muy seguros.

Albert Einstein.

Agradecimientos.

Quiero dedicar este trabajo a mi familia, porque son y siempre han sido los que más me han ayudado en todo cuanto he emprendido.

También quiero agradecer el apoyo recibido por todo el grupo de robótica de la URJC, destacando en especial a mi tutor, Jose María Cañas, por los conocimientos trasmitidos, su paciencia y el esfuerzo que ha puesto para llevar este proyecto a buen término.

A mis amigos Juanan, Miguel, Bea y Oscar; por estar siempre dándome ese soplo de aire fresco.

A mis compañeros de promoción con los que he pasado tan buenos momentos, Cesar, Ricardo e Iván; gracias por ser así.

Resumen.

Los sistemas de visión son hoy en día uno de los elementos sensoriales más atractivos para incrementar las prestaciones de los robots. Las cámaras proporcionan mucha información sobre el entorno del robot. Además, son sensores baratos debido a su utilización masiva en otras aplicaciones. Sin embargo, extraer información a partir de las imágenes capturadas es difícil y costoso, debido a la gran cantidad datos y operaciones que hay que manejar para interpretar la información contenida en ellas.

El objetivo de este proyecto es reconstruir esquemáticamente la realidad tridimensional a partir de las imágenes de un par estéreo de cámaras identificando los bordes de los objetos en 3D.

La primera parte del sistema genera una reconstrucción basada en puntos 3D que se apoya en las técnicas clásicas de emparejamiento y triangulación. Como aporte genuino se han desarrollado tres algoritmos diferentes de segmentación que agrupan esos puntos en segmentos 3D, ofreciendo una representación de la realidad mucho mas compacta. Para llevar a cabo las pruebas experimentales obviando la incertidumbre en la calibración de las cámaras se ha implementado con OpenGl un simulador visual que genera imágenes sintéticas de entornos tridimensionalmente conocidos.

Todas las soluciones aquí expuestas han sido programadas dentro de la plataforma software *jde.c* e implementadas en forma de hebras iterativas o *esquemas*. Los algoritmos desarrollados son lo suficientemente *vivaces* sobre *hardware no específico*. Además la representación conseguida tiene una precisión centimétrica.

Índice general

Índice de figuras

Capítulo 1
Introducción

En este capítulo se da una visión global del contexto del presente proyecto: Comenzando con una descripción general de la robótica, así como de otros conceptos relacionados, la visión en la robótica y la visión estereoscópica. Terminando con una breve descripción del proyecto realizado.

1.1. Robótica

La mayor parte de la gente tiene una idea de lo que es la robótica, sabe sus aplicaciones y el potencial que ésta tiene. Antiguamente la robótica no era reconocida como ciencia, es más, la palabra robot surgió mucho después del origen de los denominados autómatas, que podemos clasificar como los primeros "robots". La historia de la robótica ha estado unida a la construcción de artefactos, que trataban de materializar el deseo humano de crear seres semejantes a nosotros que nos descargasen de trabajo.

A finales del siglo XIX se presentan las primeras máquinas "robots", pero no será hasta la segunda guerra mundial cuando se realicen los primeros diseños de esta naturaleza.Con el desarrollo de las primeras computadoras digitales se produjo una considerable evolución en este campo. Así, en 1970 una serie de investigadores del Instituto de Investigación de Stanford desarrollaron Shakey (figura 1.1). Su sistema de control creaba una reproducción interna del entorno a partir de los sensores de que disponía y desde ella calculaba su movimiento [Nillson, 1971].

La idea de robot conlleva a su vez tres capacidades claramente definidas, la capacidad de moverse, la capacidad de recabar información del medio y la capacidad de razonamiento para establecer su propio comportamiento. Desde un punto de vista más formal, los robots son dispositivos compuestos de *sensores*, *actuadores* y controladores

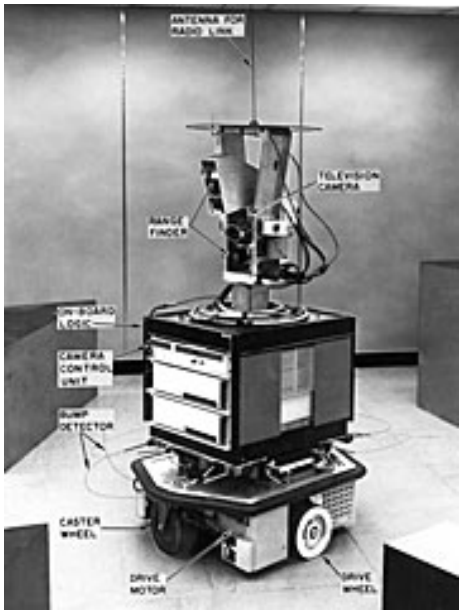
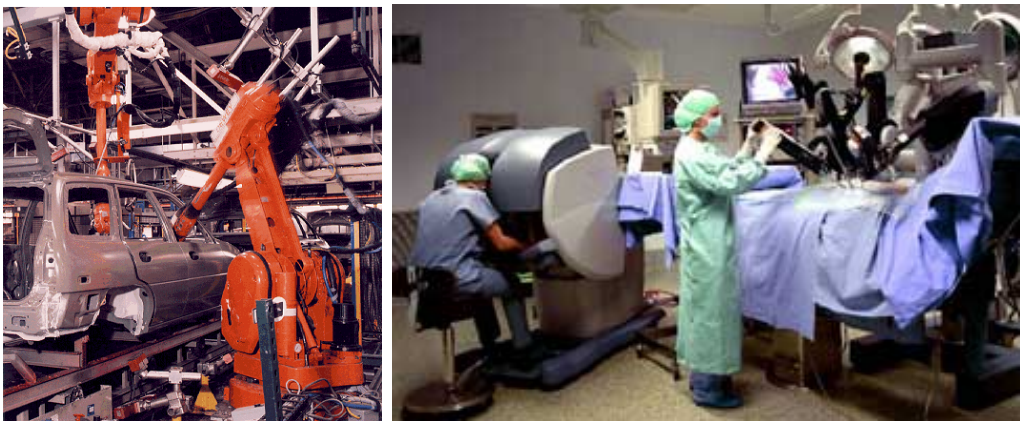


Figura 1.1: Robot Shakey.

o procesadores. Los *sensores* constituyen el sistema de percepción del robot. Son dispositivos que miden propiedades físicas como distancia, sonido, magnetismo, presión, altitud, velocidad, ... Se pueden clasificar como internos o externos. Los internos proporcionan información sobre el propio robot como puede ser su posición, velocidad o aceleración. Los externos proporcionan información sobre lo que rodea al robot como la proximidad de obstaculaos, tacto o visión.

Los *actuadores* permiten al robot interaccionar con el mundo. Son dispositivos que ejerciendo fuerzas generan movimiento en los elementos del robot. Existen varios tipos de actuadores: neumáticos (utilizan fluido compresible, aire a 5-10 bares), hidráulicos (utilizan aceite mineral a 50-100 bares) o eléctricos (por ejemplo, motores de corriente continua).

Los *controladores* y *ordenadores* son los encargados de coordinar la percepción y la actuación del robot. Necesitaremos programarlos para obtener el comportamiento deseado.Los programas que ejecutan los *ordenadores* o *controladores* que manejan al robot constituyen la inteligencia del robot, en resumidas cuentas su cerebro. Un robot sin software no tiene vida, no puede actuar ni percibir, por ello la informática está tan ligada a la robótica, ya que le da la inteligencia a los robots.



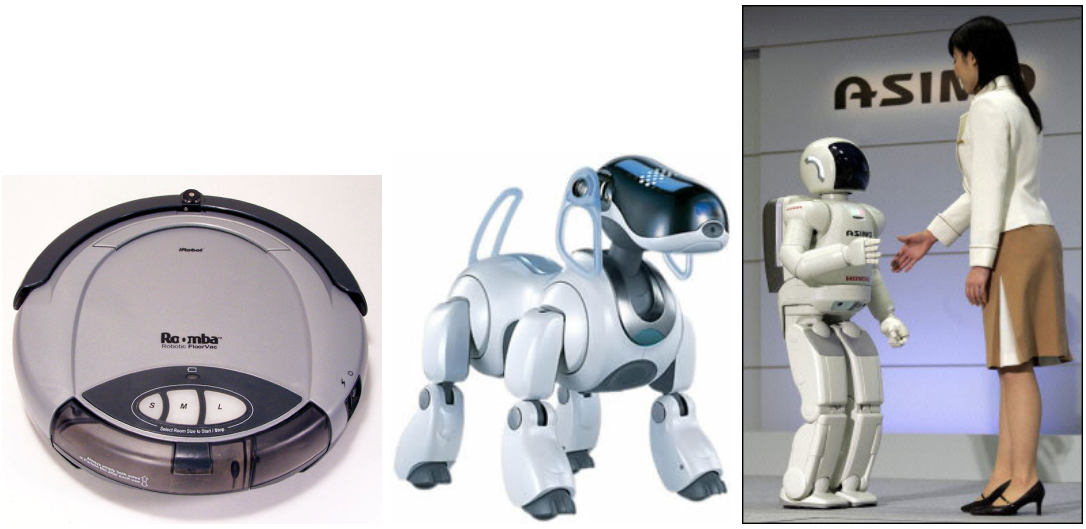
(a) Robot en cadena de producción automovilística

(b) Robot utilizado en cirugía

Figura 1.2: Aplicaciones de los robots hoy en día.

Los robots son usados hoy en día para llevar a cabo tareas peligrosas, difíciles o repetitivas para los humanos. Por ejemplo, robots articulados similares en capacidad de movimiento a un brazo humano son los más usados en las líneas de producción. Las aplicaciones incluyen soldado, pintado y carga de maquinaria. La industria automotriz ha obtenido gran beneficio de esta nueva tecnología donde los robots realizan el trabajo que antes realizaban los humanos (figura 1.2(a)).

Los robots no sólo son usados en el ámbito industrial, tienen otros muchos campos de aplicación más cercanos al público en general, como los robots que colaboran dentro de nuestras casas y oficinas realizando tareas, que van desde el simple ocio (robots mascotas, como el perrito Aibo de Sony (figura 1.3(b)), pasando por el desarrollo de tareas domésticas, como una cortadora de césped o la aspiradora Roomba (figura 1.3(a)) que ha vendido 1,5 millones de ejemplares, hasta tareas de vigilancia y seguridad. El grado de aceptación de estos robots sigue creciendo, aunque su comercialización está limitada por su elevado coste.



(a) Robot Roomba (b) Robot Aibo (c) Robot Asimo

Figura 1.3: Robots domésticos.

La medicina también hace uso de la robótica, un ejemplo de ello son los robots utilizados en procedimientos de cirugía invasiva mínima (figura 1.2(b)), robots asistenciales, telecirugía robótica o automatización de laboratorios, etc.

La exploración espacial no hubiera llegado a donde está de no haberse ayudado de la robótica, ya que los robots pueden llegar a lugares donde el hombre no es capaz de llegar, como por ejemplo Marte. Las personas necesitamos alimento, agua, y oxígeno, elementos que los robots nunca necesitarán, en definitiva los robots tienen menos necesidades, pueden resistir mejor en condiciones extremas, con la ventaja de no tener que volver a casa.

1.2. Visión en robótica

Una de las capacidades en la que los humanos se apoyan para interactuar eficientemente con el entorno es la capacidad visual. La robótica siempre la ha querido utilizar como capacidad sensorial propia de los robots. Es tan compleja que ha dado lugar a una rama científica denominada visión artificial.

El concepto de visión artificial surge en la década de los 60 con la idea básica de conectar una cámara de vídeo a una computadora. Esto implicó no sólo la captura de imágenes a través de la cámara, sino también la comprensión de lo que estas imágenes

representaban. Un resultado muy importante de este trabajo y que marcó el inicio de la visión artificial, fue un trabajo realizado por Larry Roberts, el creador de ARPAnet. En 1961 creó un programa con el que un robot podía ver una estructura de bloques sobre una mesa, analizar su contenido y reproducirla desde otra perspectiva, demostrando así que esa información visual que había sido mandada al ordenador por una cámara, había sido procesada adecuadamente por él.

Algunos ejemplos de aplicaciones actuales de la visión artificial pueden ser los reconocedores de texto *OCR* o los sistemas de seguridad basados en reconocimiento de iris (figura 1.4).

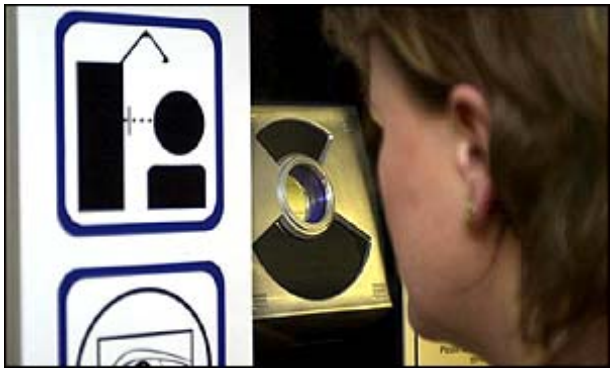


Figura 1.4: Sistema de control de acceso basado en escáner de iris.

Dentro de la robótica la visión es fundamental ya que mediante unos sensores de bajo coste se obtiene gran cantidad de información. son tantos los datos recibidos desde las cámaras que no son digeribles directamente. Para poder obtener información útil es necesario procesar las imágenes y esta tarea conlleva muchísimo cómputo. La información útil a extraer puede ser una reconstrucción de los objetos de su entorno o la localización del robot en el mundo. Podemos poner a modo de ejemplo el robot humanoide ASIMO (figura 1.3(c)) de Honda que utiliza sus cámaras para reconocer a humanos en su alrededor.

La visión artificial en robótica estudia el uso de cámaras como sensores. Su función principal es reconocer y localizar objetos en el entorno mediante el procesamiento de las imágenes. La visión artificial estudia estos procesos para construir máquinas con capacidad de “entender” una escena o las características de una imagen.

De manera natural el mecanismo de visión humano es estéreo, es decir, somos

capaces de apreciar, a través de la visión binocular, las diferentes distancias y volúmenes en el entorno que nos rodea. Nuestros ojos, debido a su separación, obtienen dos imágenes con pequeñas diferencias entre ellas, a lo que denominamos disparidad. Nuestro cerebro procesa las diferencias entre ambas imágenes y las interpreta de forma que percibimos la sensación de profundidad, lejanía o cercanía de los objetos que nos rodean. Este proceso se denomina estereopsis.

En general, la visión estereoscópica o también llamada visión en relieve, se define como la capacidad de los sistemas visuales de dar aspecto tridimensional a los objetos a partir de las imágenes en dos dimensiones obtenidas en un par estéreo.

Los avances en geometría proyectiva, pares estéreo y auto-calibración han abierto la puerta a la extracción eficiente de información tridimensional de las imágenes. Su uso permite alcanzar nuevas cotas en el área de la robótica, pues la información tridimensional permite conocer mejor y con una mayor precisión el entorno.

Gracias al impulso de estas últimas técnicas es posible, hoy día, reconocer y reconstruir rostros u objetos tridimensionales. Por ejemplo, la representación de piezas industriales, el reconocimiento de personas mediante el estudio biométrico, el diseño infográfico para la industria del cine, la programación de videojuegos con mayor interactividad, etc.

Teniendo todo esto en cuenta podemos entender el merecido interés que tiene la visión en la robótica ya que un robot dotado de percepción visual estereoscópica podrá reconocer el mundo que le rodea. Podrá, por ejemplo, localizar objetos y obstáculos, y de este modo tomar buenas decisiones en cuanto a la actuación en su entorno se refiere.

Universidades como la Carnegie-Mellon University han desarrollado aplicaciones robóticas basadas en visión. Un ejemplo es el robot MINERVA [Thrun *et al.*, 1999] (figura 1.5), que ejercía de guía para grupos de visitantes en el del Museo Nacional de Historia Americana. En este caso utilizaba una cámara enfocando al techo para ayudarse en la localización.

Dentro del grupo de robótica de la URJC también se pueden encontrar algunos ejemplos de robots que usan vision: *Seguimiento de un objeto en 3D mediante un*

Figura 1.5: Robot MINERVA para guiado autónomo de visitantes a museos.

filtro de Partículas [Barrera y Cañas, 2004], o *Navegación de un Robot con un sistema de atención visual 3D* [Cadahía, 2006]. Este ultimo proyecto permitía al robot Pioneer navegar en base a la detección tridimensional de balizas utilizando para ello estereovisión. También debemos mencionar *Comportamiento sigue pelota en un robot con visión local* [Martín, 2002], en este proyecto el robot EyeBot actúa siguiendo una pelota a una cierta distancia, para ello utiliza visión monocular o el *Comportamiento sigue persona con visión* [Herencia, 2004] que permite al robot Pioneer perseguir a un humano basándose en las imágenes obtenidas por una *webcam*.

1.3. Reconstrucción esquemática 3D desde visión estéreo

El presente proyecto tiene como objetivo conseguir una representación esquemática tridimensional de la escena desde un par estéreo de cámaras. En definitiva, localizar en 3D los bordes de la escena, basándose solamente en los estímulos visuales de un par estéreo, es decir, reconstruir una realidad sintetizada del entorno tridimensional que rodea al robot sobre el que están las cámaras.

Este trabajo constituye la evolución de la tesis doctoral *Murphy: hacia un robot con visión estereoscópica* [P.Bustos, 2003] aportando nuevas características como el simulador visual o la segmentación de la representación tridimensional.

En el contexto cercano de este proyecto podemos encontrar trabajos como *Reconstrucción 3D visual con algoritmos evolutivos* [Martínez, 2007]. En este proyecto, se utilizó un par estéreo con el objetivo de poder reconstruir la escena utilizando algoritmos genéticos. Tambien el poyecto *Representación rica de la escena 3D alrededor de un robot móvil* [Maya, 2006], que entre uno de sus objetivos tenía la detección tridimensional de puertas, paredes y personas, utilizando para ello las percepciones sensoriales de una cámara y un sensor láser. Todos estos trabajos, incluido éste, tienen como objetivo sintetizar la realidad tridimensional de la manera más compacta posible para que de este modo sea manejable, lo que implica ser útil. Sin embargo, la extracción de esta información útil a partir de las imágenes proporcionadas por las cámaras es una tarea compleja.

La memoria se organiza como sigue: En el siguiente capítulo se describe cuál es el objetivo concreto del proyecto y qué requisitos se deben cumplir. En el tercero se describen las herramientas software que se han usado para su implementación, tanto hardware como software. En el capítulo cuarto se explica el sistema desarrollado, hablando tanto del diseño como de la programación del mismo. En el capítulo quinto se comentan las pruebas experimentales realizadas y en el último capítulo se recogen las conclusiones obtenidas así como las mejoras que se puedan efectuar para trabajos futuros.

Capítulo 2
Objetivos

Una vez presentado el contexto general y particular de este proyecto, vamos a fijar los objetivos concretos y los requisitos que han condicionado su desarrollo.

2.1. Descripción del problema

El objetivo del proyecto es la *reconstrucción tridimensional 3D visual con triangulación* y su segmentación en tramos rectos. Con este propósito, se ha diseñado e implementado una aplicación que captura la información de un par estéreo de cámaras y la interpreta utilizando diferentes algoritmos para en ultimo termino represente los resultados obtenidos en 3D.

El objetivo final lo hemos articulado en los siguientes subobjetivos:

1. *Implementación de un simulador visual de escenas 3D.* Es necesario contar con una herramienta que provea de imágenes simuladas de entornos conocidos para poder depurar y probar los algoritmos de reconstrucción.

2. *Diseño e implementación de algoritmo de reconstrucción 3D con puntos.* Una vez filtradas las imágenes debemos interpretar lo que hemos detectado en ellas. Para ello se aplicarán diferentes algoritmos clásicos:

   a) *Algoritmo de selección de homólogos.* Se establecen las correspondencias entre los puntos interesantes de una imagen con sus homólogos en la otra imagen del par [Mühlmann *et al.*, 2001].

   b) *Triangulación.* Con un punto y su homólogo calcula la posición que le corresponde en 3D. Estos puntos 3D son la primitiva de la reconstrucción [J.M López Valles, 2005].

3. *Diseño e implementación de algoritmos de reconstrucción 3D con segmentos.* La representación 3D basada en puntos es poco compacta, por lo que es necesario generar una representación más compacta basada en segmentos 3D. Se diseñará por completo un algoritmo propio destinado a esta tarea.

4. *Experimentos.* Realización de pruebas para comprobar la bondad de los algoritmos implementados, así mismo también se caracterizan las deficiencias detectadas y se realizará una valoración empírica.

2.2. Requisitos

El desarrollo de este proyecto está guiado por los objetivos comentados anteriormente y deberá ajustarse a los requisitos que ahora describimos:

- Implementación del sistema dentro de la *plataforma software jde.c*[1].
- Funcionamiento del sistema con *hardware convencional* no específico.
- Precisión centimétrica en la reconstrución.
- Procesamiento de imágenes con *dinamismo*, vivacidad de los algoritmos.

2.3. Metodología y plan de trabajo

El plan de trabajo utilizado en la realización de este proyecto ha consistido en el *modelo de desarrollo en espiral basado en prototipos* (figura 2.1). Este modelo de desarrollo se basa en la realización de varias subtareas sencillas que conjuntamente compondrán el comportamiento final del sistema. Usando este modelo se aporta cierta flexibilidad en cuanto a posibles cambios de requisitos.

Este modelo de desarrollo se caracteriza por la relación de las subtareas en un número determinado de ciclos. En cada uno de estos ciclos existen cuatro etapas: *Análisis de requisitos, Diseño e Implementación, Pruebas y Planificación del próximo ciclo de desarrollo.*

[1] https://trac.robotica-urjc.es/jde

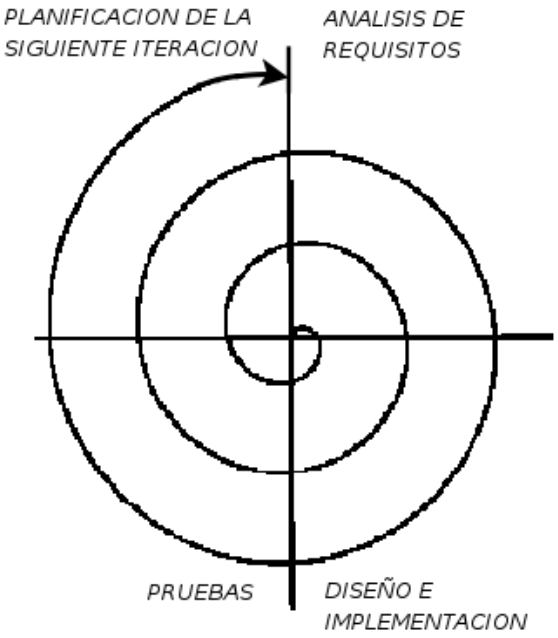


Figura 2.1: Modelo en espiral

Para el desarrollo de este proyecto se han completado los siguientes ciclos:

1. *Formación y familiarización con la infraestructura:*

   a) *Entorno Linux y sus herramientas básicas.* Instalación de paquetes y librerías en la distribución Ubuntu.

   b) *Plataforma software jde.c* [Plaza, 2004] estudio a fondo de la estructura de la misma. La primera iteración de la espiral consistió en la creación de varios esquemas para comprender el funcionamiento general de jde.c.

   c) *Lenguaje C y xforms.* Durante todas las etapas iniciales del proyecto se estudió exhaustivamente el lenguaje C, así como la biblioteca gráfica xForms.

2. *Estudio de la biblioteca OpenGL* utilizada para la simulación de percepciones visuales y para la visualización de los entornos reconstruidos.

3. *Estudio de técnicas de visión artificial*, estereovisión, correlación, disparidad, triangulación, filtros de bordes, esquinas, etc.

4. *Diseño e implementación del simulador 3D.* En esta etapa de la espiral se implementó un simulador visual utilizando la biblioteca OpenGL.

5. *Implementación de algoritmo de reconstrucción basado en puntos.* En esta fase, se estudiaron las alternativas existentes sobre algoritmos de reconstrucción puntual.

Se fueron probando diferentes alternativas hasta obtener una implementación vivaz.

6. *Diseño e implementación de algoritmos de reconstrucción basados en segmentos.* En esta fase, se diseñaron diferentes alternativas al proceso de reconstrucción basado en segmentos. Se probaron diferentes aproximaciones lo que dio como resultado tres algoritmos genuinos de reconstrucción basada en segmentos.

Durante todo el desarrollo del proyecto se han mantenido reuniones semanales con el tutor para establecer y pulir los puntos que se han llevado a cabo en cada etapa.

Capítulo 3
Entorno y plataforma de desarrollo

Antes de abordar la materializacón concreta del proyecto, en este capítulo vamos a describir la plataforma y bibliotecas en las que nos hemos apoyado para su implementación.

3.1. Plataforma jde.c

La plataforma jde.c[1] es una implementación de la arquitectura cognitiva ***J****erarquía* ***D****inámica de* ***E****squemas* (***JDE***) [Plaza, 2003]. Es un entorno software para la programación de robots móviles y ha sido diseñado por el Grupo de Robótica de la URJC. Este entorno, al igual que otras plataformas de desarrollo en programación de robots, facilita la creación de programas para que el robot se comporte de una determinada manera y exhiba conductas autónomas. Para ello resuelve los aspectos más generales de la programación de robots, como son el acceso a los sensores y actuadores, la multitarea, las interfaces gráficas y las comunicaciones entre programas. En la práctica incluye tres servidores, varias librerías y un conjunto de ejemplos cuyo código puede servir de base para nuevas aplicaciones.

En terminos generales la arquitectura JDE se basa en pequeñas unidades de comportamiento denominadas *esquemas*. Estos esquemas son flujos de ejecución independientes (modulables, iterativos que pueden ser activados o desactivados a voluntad) con un determinado objetivo. En cada uno de ellos se encapsula la diferente funcionalidad que podrá ser reutilizada en posteriores ocasiones.

Existen dos tipos de esquemas en JDE, los esquemas perceptivos y los esquemas motores o de actuación. Los esquemas perceptivos son los encargados de recoger

[1] https://trac.robotica-urjc.es/jde

los datos sensoriales, procesarlos y poner esa información a disposición de cualquier esquema. Los esquemas motores utilizan la información proporcionada por los esquemas perceptivos para generar las órdenes de movimiento sobre el robot. Este proyecto está claramente definido como un esquema perceptivo ya que recoge información sensorial proveniente de las cámaras para elaborar con ella una construcción tridimensional sobre el entorno.

3.2. Biblioteca Progeo

En conjunción con la plataforma de desarrollo *jde.c* se han utilizado también tres bibliotecas adicionales en este proyecto fin de carrera. La primera de ellas es *biblioteca de geometría proyectiva Progeo* utilizada para operar con un espacio tridimensional. Esta biblioteca es utilizada en la aplicación para relacionar el mundo de las imágenes (2D) con el mundo real (3D).

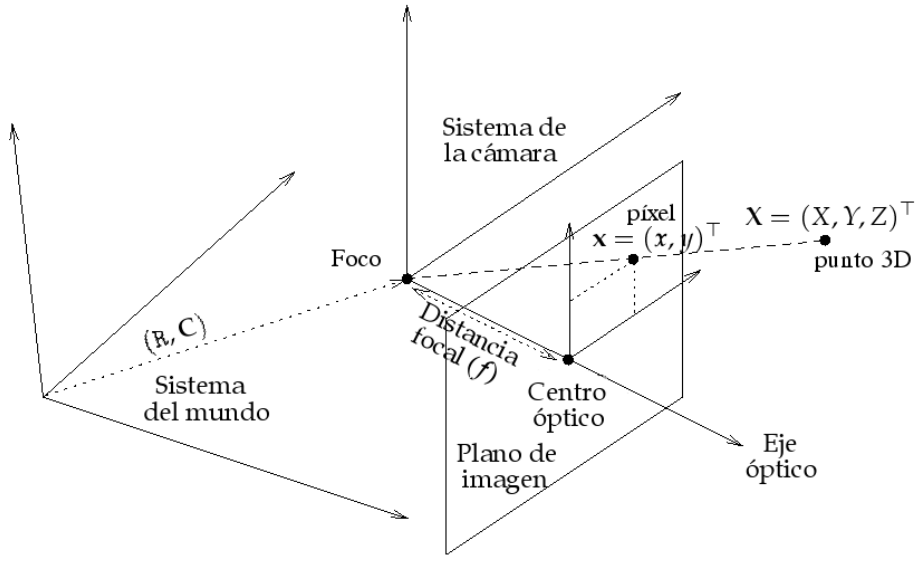


Figura 3.1: Modelo de cámara Pinhole

Esta relación se consigue gracias a dos funciones principales:

- *Proyectar*. Esta función permite realizar la proyección geométrica de un punto 3D del espacio, en el plano imagen de la cámara, obteniendo así un punto 2D perteneciente a ese plano. Ese punto 2D nos da información del píxel de la imagen en el que proyecta el punto 3D correspondiente.
- *Retroproyectar*. Esta función permite obtener la recta en el espacio 3D formada por todos los puntos que proyectan en un píxel determinado. Devuelve como resultado las coordenadas 3D de un punto arbitrario de dicha recta, que pasa tambien por el focp de la cámara.

Progeo se basa en el modelo de cámara *pinhole* (figura 3.1) [Hartley y Zisserman, 2004]. Para realizar estas transformaciones la biblioteca necesita conocer los parámetros de las cámaras, tanto intrínsecos como extrínsecos. De otro modo no proporciona resultados fiables. En este proyecto utilizamos camaras sinteticas de las cuales conocemos exactamente el valor de estos parámetros.

3.3. Biblioteca Susan

Para poder llevar a cabo el filtrado de las imágenes se ha utilizado la biblioteca *susan*[2](Smallest Univalue Segment Assimilating Nucleu). Esta biblioteca permite funcionalidades como la detección de bordes o esquinas de una imagen. Las funciones de extracción de bordes y esquinas de esta biblioteca se basan en determinar las áreas que tienen brillo uniforme, considerando como borde la frontera que separa estas áreas y esquinas los puntos encerrados entre ellas.

Las funciones utilizadas de esta biblioteca han sido *i2edges* para bordes y *i2corners* para la detección de esquinas. Dichas funciones reciben como entrada imágenes en escala de grises y ofrece como salida imágenes en escala de grises con los píxeles detectados en color blanco. El interfaz de salida de las funciones mencionadas implica una exploración píxel a píxel para obtener las coordenadas 2d de cada punto detectado. Por ello se ha modificado la biblioteca para que devuelva como salida una estructura de datos con la información correspondiente a la colección de puntos detectados. De

[2]http://www.fmrib.ox.ac.uk/steve/susan/index.html

esta manera, la obtención de la información relevante es inmediata.

El filtrado de bordes es fundamental en la tarea que aborda este proyecto, ya que intentar una reconstrucción 3D sobre todos los pixeles de las imágenes es inviable si queremos obtener una respuesta vivaz. Para limitar el conjunto de píxeles a colocar en 3D utilizamos el filtro de bordes y esquinas ya que son las zonas que definen a los elementos relevantes de la escena.

3.4. Biblioteca OpenGL

OpenGl[3] ofrece un API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D. Esta especificación fue desarrollada por Silicon Graphics Inc[4] y a partir de ella, los fabricantes han creado hardware específico que le permite dar apoyo. Las operaciones necesarias para la materialización de estos gráficos se delegan en la GPU (procesador de la tarjeta gráfica). De este modo se libera a la CPU de las tareas propias de la generación de gráficos quedando disponible para cualquier otra tarea.

Esta biblioteca se ha utilizado para la simulación visual, empleado *renderizado offscreen* que permite obtener imágenes (2D) en RGB de escenas (3D) modeladas con las primitivas de OpenGL. También ha sido empleada en el proceso de visualización de la reconstrucción 3D, ya que permite generar escenas tridimensionales cómodamente, ofreciendo diferentes puntos de vista y relaciones de aspecto.

[3]http://www.opengl.org
[4]http://www.sgi.com/global/es

Capítulo 4
Descripción informática

En este capítulo detallamos la solución desarrollada para alcanzar los objetivos propuestos en el capítulo 2. En primer término introduciremos un esbozo de la solución y posteriormente describiremos en detalle cada una de sus partes.

4.1. Diseño general

La aplicación se descompone en dos esquemas independientes de la arquitectura jde: *Simulador Visual*, encargado de generar percepciones visuales sintéticas y *Murphy*, dedicado a generar la reconstrucción 3D a partir de las imágenes de dos cámaras (en este caso sintéticas).

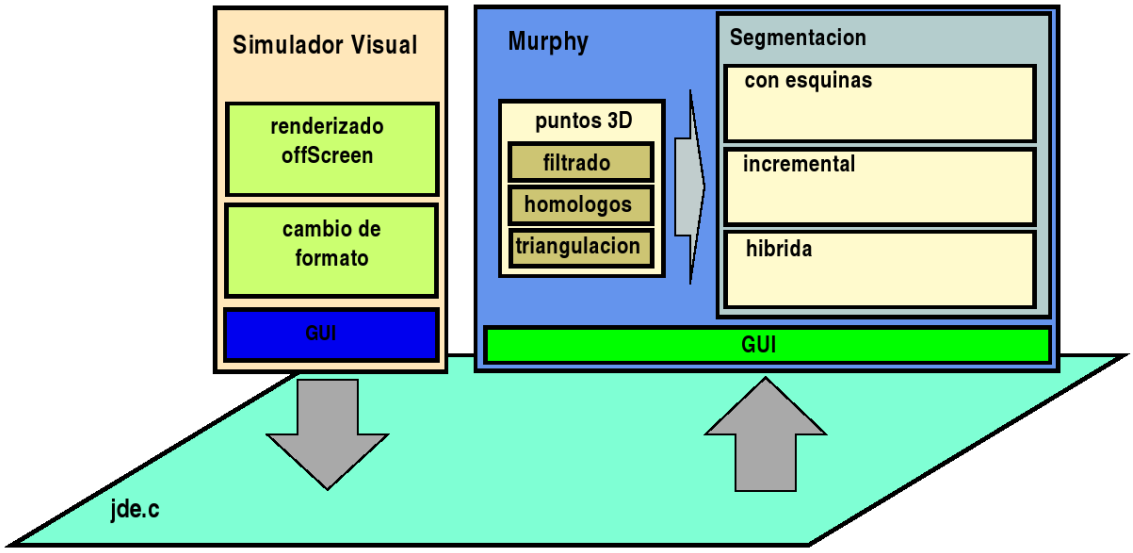


Figura 4.1: Diseño general de la solución.

4.2. Simulación visual

El presente proyecto permite, como anteriormente hemos mencionado, la reconstrucción 3D de entornos captados a través de un par estéreo de cámaras. Para llevar a cabo esta tarea es indispensable proveer a los algoritmos de reconstrucción con imágenes de entrada, y con ellas los parámetros asociados a las cámaras que las generaron. Intentar abordar la depuración y pruebas de los algoritmos de reconstrucción mediante imágenes captadas por cámaras reales resulta complicado, ya que la caracterización de sus parámetros no es perfecta, lo que desemboca en errores de reconstrucción. Este fenómeno se produce por el alto grado de dependencia en los algoritmos de reconstrucción a la calibración y caracterización de las cámaras.

Se pretenden consegir algoritmos de reconstrucción fiables. Por lo que para analizar su funcionamiento y aislar los problemas de calibración los usaremos con imagenes generadas con camaras sinteticas perfectamente caracterizadas. De modo que la calidad de la representación dependera exclusivamente del algoritmo reconstructivo y no de las imprecisiones de la calibración. Es entonces inevitable elavorar un simulador visual que nos permita generar imágenes simuladas construidas a partir de camaras sinteticas en las que se puedan establecer todos sus parámetros, con lo que se eliminan los errores de calibración.

Para generar las imagenes sinteticas utilizaremos OpenGl, por lo que hubo que familiarizarse con el modelo de cámara utilizado por la biblioteca. Otra característica indispensable es la transparencia en cuanto al origen de las imágenes de cara al esquema encargado de la reconstrucción. Para conseguir esto es indispensable que el simulador visual ofrezca las imágenes sintéticas de la misma manera que la plataforma jde.c ofrece las imágenes reales, en concreto en forma de variable compartida.

El formato utilizado por jde.c son imágenes de 320*240 en rgb con un byte para cada color; en definitiva, un array de 320*240*3 bytes comenzando por el píxel de la esquina inferior izquierda de la imagen. El uso típico de OpenGl es el del renderizado y visualización de escenas 3D, no el de la generación de imágenes sinteticas. Indagando en la especificación del API de OpenGl, descubrimos que es posible realizar esta tarea con lo que se denomina *renderizado offsreen*.

El *renderizado offsreen* consiste en renderizar escenas pero no para su visualización directa sino fuera del área de visualización. OpenGl necesita para ello que se le establezca una zona de memoria para guardar el estado (OpenGl funciona como una máquina de estados) de la renderización *(rendering context)*. De ese contexto de *renderización* se puede extraer la colección de píxeles asociados con la vista de esa escena, es decir, la imagen que genera a nivel de píxeles. Para obtener dicha colección de píxeles es necesario invocar *glReadPixels()*, que devuelve la colección de píxeles asociados al contexto activo antes de su invocación. El resultado que ofrece la llamada a *glReadPixels()* es justo la imagen sintética que necesitamos generar salvo por una peculiaridad, la colección de píxeles que retorna está ordenada al revés en cuanto a filas se refiere, comenzando por la esquina superior izquierda. Es necesario entonces cambiar el orden antes de hacer disponible la imagen a la plataforma. Para hacer visible esta variable al resto de esquemas, la arquitectura jde.c cuenta con las funciones *myexport* y *myimport* que permiten poder compartir variables entre esquemas.

El API de OpenGl define cómo establecer las relaciones de aspecto y el tipo de proyección de las cámaras sintéticas que utiliza para el renderizado, es decir, sus parámetros. Como usaremos la biblioteca Progeo en el proceso de reconstrucción para relacionar las imágenes 2d con el mundo 3D, es necesario encontrar la relación entre el modo de especificar los parámetros de las cámaras entre ambas bibliotecas y de este modo especificar las cámaras de OpenGl al estilo de Progeo. Por suerte, las dos bibliotecas utilizan el modelo de cámara *pinhole* [Hartley y Zisserman, 2004], por lo que únicamente difieren en la forma de especificar dichos parámetros.

Los extrinsecos son expresados en ambas biblotecas como la posición de la cámara en el mundo, el foco de atención y la orientación o roll. Sin embargo, los parametros intrinsecos son caracterizados en cada biblioteca de manera diferente. Porgeo necesita la distancia focal y la posición del centro del plano imágen,y por el contrario, OpenGl utiliza el angulo de apertura y la relación de aspecto del plano imágen. La equivalencia existente entre ambas se determina facilmente ya que el tamaño de las imágenes es conocido (320*240 relacion de aspecto), el centro óptico de la cámara OpenGl es 120,260 (al tratarse de una cámara ideal) y el ángulo de apertura equivale a $\alpha = 2 * \arctan \frac{\frac{alto}{2}}{f}$. Tenemos determinada la equivalencia entre las diferentes formas de expresar los parámetros intrínsecos que tiene cada bibloteca.

Todos los parámetros de las cámaras utilizados por el simulador son exportados a la

arquitectura en formato progeo para que sean accesibles por los esquemas que quieran utilizar la simulación visual.

El API que ofrece el simulador implementado es el siguiente:

- *simuladaA:* variable equivalente a colorA[1] pero con la imagen sintética.
- *simuladaB:* variable equivalente a colorB[2] pero con la imagen sintética.
- *camaraA:* variable que contiene los parámetros de la cámara usada en la construcción de la imagen simuladaA al estilo Progeo.
- *camaraB:* variable que contiene los parametros de la cámara usada en la construcción de la imagen simuladaB al estilo Progeo.

Para dar generalidad al simulador en su GUI (Graphics User Interface), se permite configurar las cámaras en diferentes modos, permitiendo así trabajar con configuraciones independientes o como un par estéreo. El simulador permite generar escenas simples con varios cubos estáticos o en movimiento.

Teniendo en cuenta el API proporcionado, los esquemas acceden de la misma manera a las imágenes reales o a las sintéticas, simplemente leyendo en cada caso de las variables adecuadas.

4.3. Reconstrucción puntual

La reconstrucción 3D con triangulación [J.M López Valles, 2005] es una técnica bastante experimentada, de la cual se obtienen buenos resultados. Ligada a ella se asocian técnicas para la detección de puntos homólogos [S. Birchfield, 1999]. Se ha implementado la solución clasica añadiendo algunas mejoras en la correlación.

El proceso de reconstrucción puntual se ha materializa en 3 etapas:

- Filtrado de imágenes
- Detección de homólogos

[1] Variable en la que la arquitectura jde.c coloca la imagen izquierda
[2] Variable en la que la arquitectura jde.c coloca la imagen derecha

- Triangulación

Ahora pasamos a detallarlas pormenorizadamente, haciedo énfasis en el cómo de su funcionamiento.

4.3.1. Filtrado de imágenes

Para llevar a cabo de la mejor manera posible la reconstrucción de la escena, debemos seleccionar de alguna manera los puntos interesantes de las imágenes, ya que intentar reconstruir todos los píxeles de la escena a tiempo real es inmanejable. Para esta tarea hemos decidido utilizar un filtro de bordes y otro de esquinas. En nuestro caso, los proporcionados por la biblioteca susan. Los bordes (figura 4.2) y esquinas sintetizan bastante bien cualquier escena.

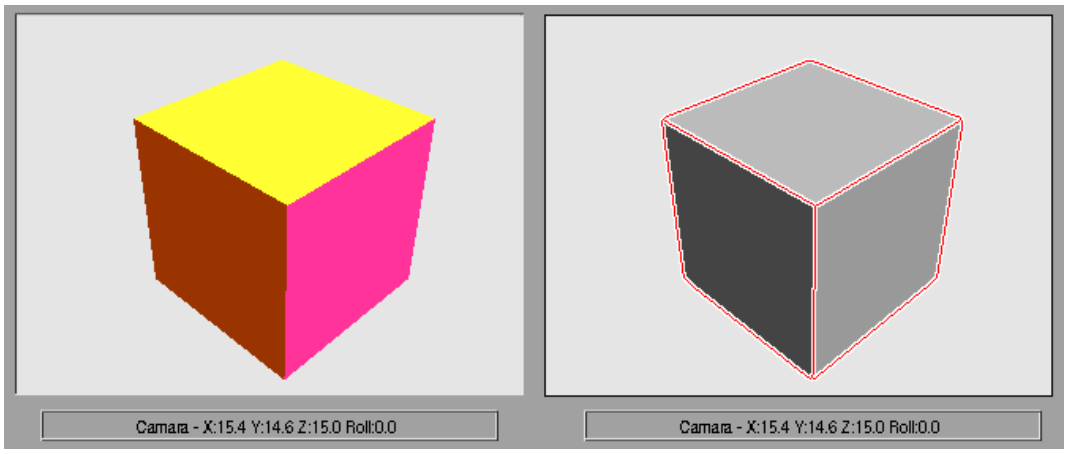


Figura 4.2: Resultado de la aplicación de un filtro de bordes.

Los puntos detectados bajo estas características serán los utilizados como base por los algoritmos de segementación.

4.3.2. Calculo de homólogos

Cualquier algoritmo de reconstrucción 3D que se base en triangulación necesita conocer la posición de las proyecciones de un punto 3D en cada cámara del par. Con el proceso de filtrado tenemos los conjuntos de puntos interesantes de cada imagen pero no sabemos qué punto interesante de la imagen izquierda coincide con cuál otro de la imagen derecha (figura 4.3).

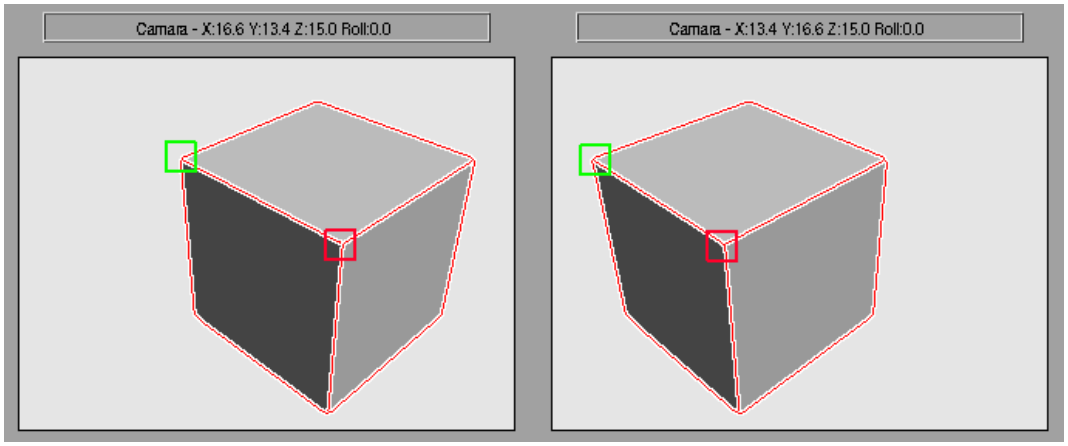


Figura 4.3: Ejemplo de puntos homólogos entre las imágenes del par.

Para emparejar cada píxel con su homologo utilizaremos la técnica de la correlación [S. Birchfield, 1999]. La correlación consiste en coger un entorno cercano al píxel a emparejar (su vecindad o ventana de correlación), y buscar en la imagen homóloga el mejor encaje a esa ventana entre los puntos interesantes (puntos borde) de la imágen contraria (figura 4.4). Comparandose las vencindades del pixel a emparejar con todos los posibles candidatos. En nuestro caso se toma un tamaño de ventana de 5*5 pixeles.

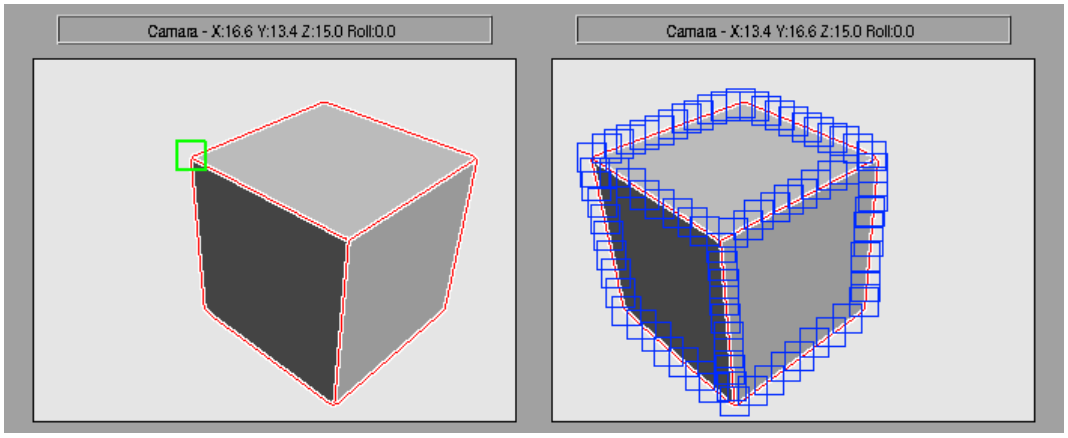


Figura 4.4: Correlación entre todo los puntos interesantes.

Realizar la correlación sobre todas los puntos interesantes (puntos borde) de la imágen homóloga requiere mucho tiempo de computación. Para mejorar el tiempo de respuesta de la correlación añadimos la restricción epipolar, que consiste en limitar la búsqueda sobre el epípolo de la imagen homóloga. El epipolo es el resultado de la proyección en la cámara homologa de la recta de proyección que causó el píxel a emparejar (figura 4.5).

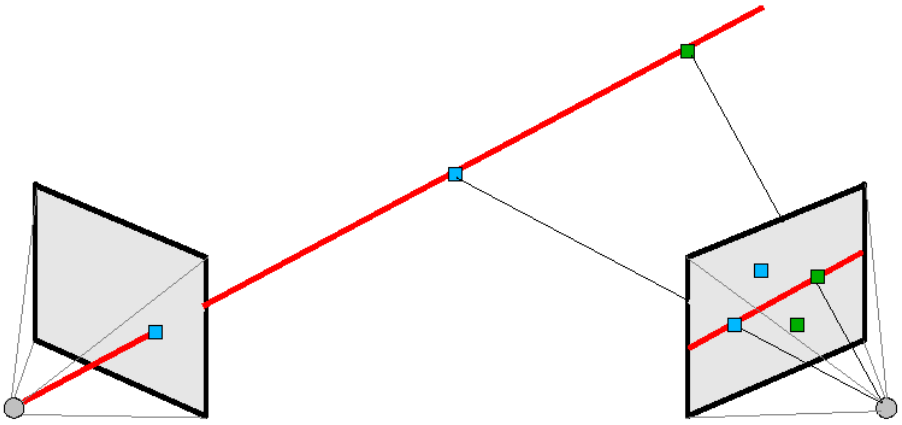


Figura 4.5: Restricción epipolar.

Como vemos en la (figura 4.6), el píxel que ha provocado la proyección en la cámara izquierda produce necesariamente su proyección en la cámara derecha a lo largo del epipolo.

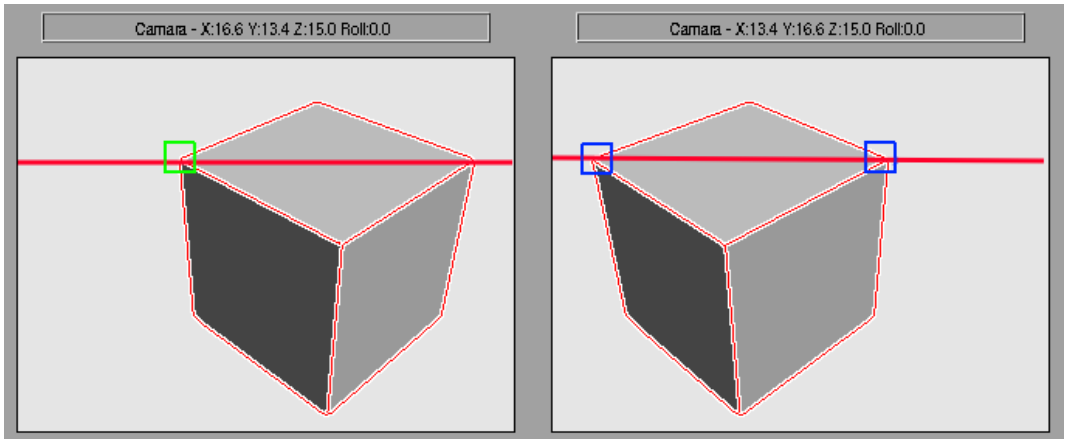


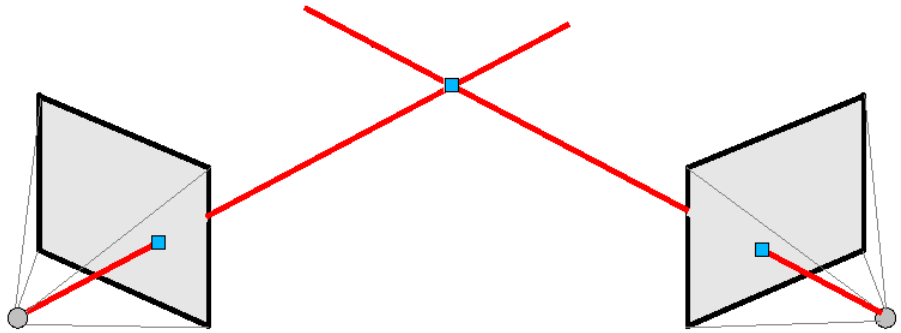
Figura 4.6: Correlación entre los puntos interesantes de la epipolar.

Aun dentro de la linea epipolar en muchas ocasiones no tenemos un encaje que destaque por su mayor parecido, surge ambigüedad a la hora de decidir qué encaje es el mejor, ya que aparecen varias ventanas con el menor grado de diferencia. Este fenomeno se produce en imágenes homogeneas como es el caso de las sinteticas. Se preve que este fenomeno se acuse en menor medida con imágenes reales dado su caracter irregualar. En cualquier caso se ha diseñado una heuristica que resuelve este problema.Consiste en calcular la correlación de los píxeles adyacentes a un píxel en su propio epípolo, dando como resultado un perfil del parecido que tiene con su alrededor. Realizando este proceso entre los diferentes píxeles a desambiguar y el pixel original se escoque

el que tiene un perfil que se asemaja mas con el del pixel original.Esta heuristica no compara las diferencias entre las ventanas sino las diferencias del parecido del píxel con su entorno.

4.3.3. Triangulación

Con la colección de pares homólogos conseguida por el algoritmo de correlación, se dispone de las proyecciones de un mismo punto 3D desde diferentes puntos de vista. Las rectas de retroproyección de cada cámara se cortan en la posición asociada a ese punto 3D (figura 4.7(a)).



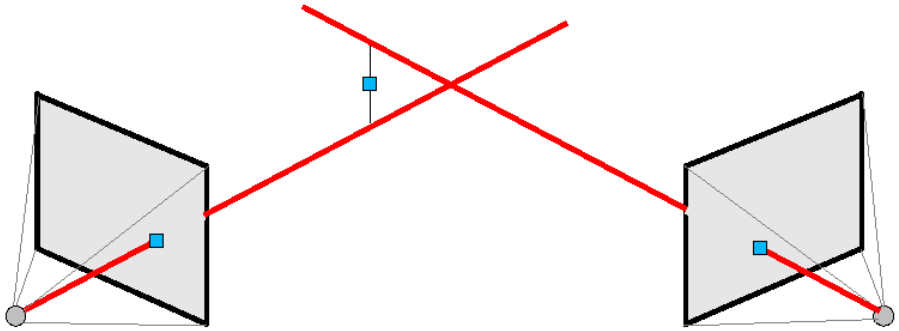
(a) Triangulación

Figura 4.7: Triangulación ideal

Conocida la posición 3D del foco de la cámara y apoyándonos en la función *backProject* de Progeo, que a partir de un punto proyectado y los parámetros de la cámara devuelve un punto 3D contenido en la recta de retroproyección (recta 3D que desemboca en ese punto 2D), podemos definir la recta de retroproyección asociada a cada cámara. Resolviendo el sistema de ecuaciones que se plantea con las dos rectas obtenemos la posición 3D correspondiente a ese par de píxeles homólogos [J.M López Valles, 2005].

Como anteriormente comentamos, los parámetros de cámaras reales puede que no sean conocidos con exactitud, produciendo que las rectas obtenidas como hemos descrito no se corten. Para dar robustez a la aplicación ante este problema, relajamos la idea de corte a cruce. Es evidente que necesariamente se tienen que cruzar, por lo que ahora definimos el punto 3D asociado, con el punto medio del segmento de distancia mínima que une ambas rectas (figura 4.8(a)). Lógicamente, si los parámetros de las cámaras son los exactos, el resultado obtenido es el real pero, por el contrario, si eso

no sucede obtenemos una aproximación muy buena. En el capitulo 5 se mostraran reconstruciones consegudas con esta tecnica.



(a) Triangulación

Figura 4.8: Triangulación basada en segmento de distancia mínima

Realizando este procedimiento con todos los pares de puntos detectados como relevantes por los filtros de bordes y esquinas, conseguimos una colección de puntos los suficientemente densa como para reconstruir la escena.

4.4. Segmentación

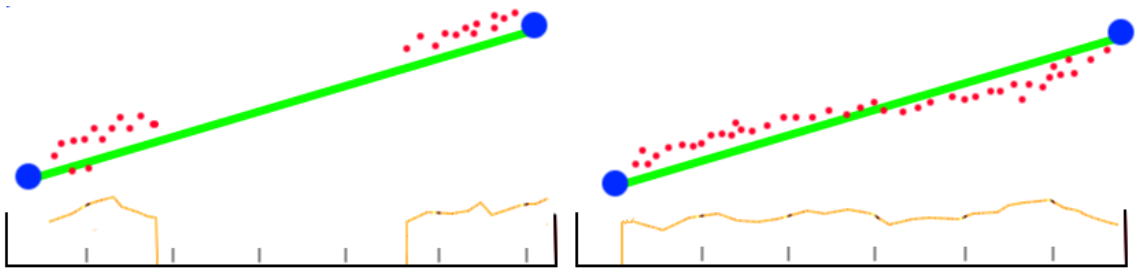
La reconstrucción 3D puntual conseguida con el algoritmo de triangulación resulta ser poco compacta. Por ejemplo, una escena con un cubo de un metro cúbico es representado con una colección de puntos del orden del millar. Por ello se han implementado diferentes algoritmos que permiten representar la reconstrucción con segmentos 3D. En un caso ideal el cubo se definiría por nueve segmentos 3D.

Se realizaron pruebas de localización de segmentos 2D basándonos en la solución propuesta en el *IV Workshop de Agentes Físicos WAF-2003* [Victor M. Gómez, 2003] para luego ser extrapolados a 3D aunque no se obtuvieron resultados óptimos. Decidimos entonces aprovechar la colección de puntos 3D de la reconstrucción puntual como punto de partida del algoritmo de segmentación.

Los algoritmos implementados se basan en la idea de validación de hipótesis de segmentos 3D, es decir, partiendo de un segmento 3D hipotético se valida si éste puede ser formado con los puntos 3D de soporte. Ahora se nos presenta el siguiente problema: ¿Cómo considerar que un segmneto hipotetico sé esta soportado por la base de puntos y

otro no?. En una priemera aproximacion consideraremos como segmentos válidos todos aquellos segmentos que tengan a su alrededor un determinado numero de puntos. Esta aproximación a primera vista puede resultar bastante buena pero, por el contrario, no daba buenos resultados. Téngase en cuenta que en el caso en el que un segmento tuviese en uno de sus extremos o a lo largo del mismo una acumulación de puntos cercanos lo suficientemente elevada, haría validar un segmento que en realidad no tiene puntos cercanos a lo largo de toda su extensión(figura 4.9(a)).

Para solventar esta problemática se ha implementado en segunda instancia la validación mediante densidad de cercanía de puntos por unidad de longitud, de este modo sólo se validan los segmentos que a lo largo de toda su extensión es decir por cada unidad de medida tienen una cantidad de puntos 3D cercanos razonable (figura 4.9(a)).



(a) Segmentos

Figura 4.9: Segmentos a validar

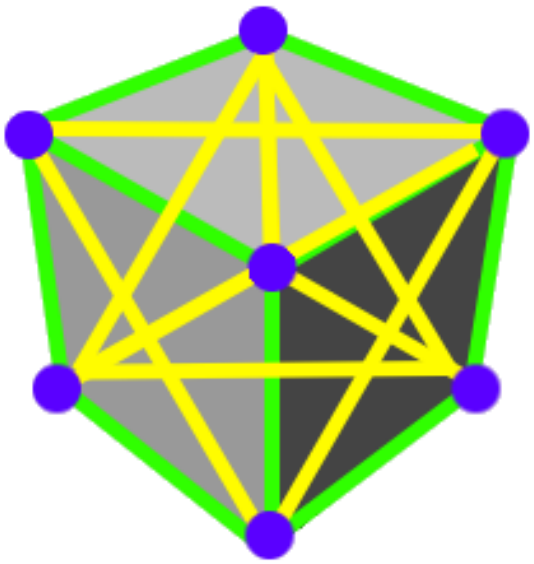
Con este mecanismo se eliminan los falsos positivos que generaba la primera aproximación. En resumen, sólo se validan los segmentos hipotetizados que tienen soporte por puntos 3D a lo largo de toda su extensión.

Para poder aplicar esta solución es necesario generar las hipótesis a validar. Es por esto que se desarrollaron tres algoritmos, utilizando cada uno de ellos un método diferente para la generación de hipótesis. Estos algoritmos se describirán en detalle a continuación.

4.4.1. Segmentación basada en esquinas

En el proceso de reconstrucción puntual se tomaban como primitivas los bordes y esquinas de los objetos. Este algoritmo hace uso de las esquinas 3D detectadas para

combinarlas y obtener una colección de segmentos hipotéticos prometedores. Puesto que las esquinas que aparecen en una escena tienen entre sí una posibilidad bastante elevada de formar un borde, es decir, ser un segmento 3D (figura 4.10(a)).



(a) Cubo

Figura 4.10: Segmentos posibles entre la esquinas de un cubo

Utilizando este conjunto de hipótesis y aplicando el criterio de validación de segmentos 3D descrito anteriormente, se obtiene como resultado el conjunto de segmentos 3D que conforman la escena. Este algoritmo cuenta con la ventaja de que genera muy pocas hipótesis para, previsiblemente, segmentar toda la representación.

4.4.2. Segmentación incremental

Este algoritmo pretende segmentar la colección de puntos 3D pero sin tener dependencia de la identificación de esquinas. En el algoritmo basado en esquinas teníamos un conjunto reducido de hipótesis. Sin embargo, en este escenario no tenemos ninguna que en principio pueda ser prometedora. Siempre cabe la posibilidad de intentar hipotetizar entre todos los puntos de la colección, pero ya vimos que el orden de puntos es del millar así que la combinación de todos con todos sería del orden de millones. Computar tantas combinaciones y tener una respuesta vivaz no son compatibles.

Teniendo en cuenta este problema, convenimos en buscar alguna característica que

nos ayude a reducir el numero de posibilidades. Una de ellas es seleccionar entre las parejas de puntos que tengan una separación espacial razonable, ya que intentar hacer segmentos por debajo de la precisión de la reconstrucción es perder el tiempo, pues se computarán todos los puntos y se concluirá que el segmento no es válido. Esta característica no parecía suficiente, y de hecho el tiempo de cómputo no mejoró de manera notable.

En esta situación nos dimos cuenta de algo evidente, *hagamos que el tamaño del problema se reduzca* y con ello converja rápidamente. Un segmento que tiene soporte por puntos 3D tiene una colección de puntos que son los que le han dado soporte , si eliminamos esos puntos soporte de la colección inicial reducimos considerablemente el número de combinaciones que quedan por hacer. Cada vez que se encuentre un segmento validado reducimos el tamaño del problema consiguiendo que concluya rápidamente.

4.4.3. Segmentación híbrida

Vistos los algoritmos antes descritos pensamos que sería una buena idea combinar ambas soluciones. Esto dio lugar a un algoritmo híbrido que pretende aprovecharse de las bondades de ambos.

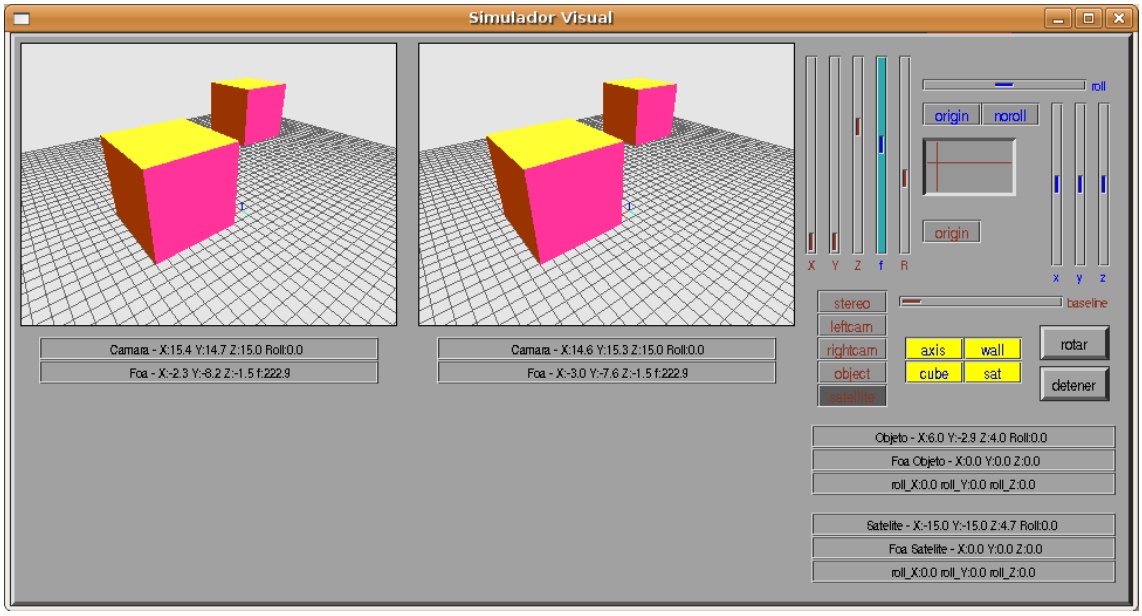
El funcionamiento de este algoritmo se basa en dos etapas. Primero se hipotetiza entre las diferentes esquinas detectadas, se validan los segmentos y se borran sus puntos soporte de la colección de cada uno que ha sido validado. Si después de este proceso el numero de puntos 3D de la colección que no ha sido capaz de "explicar" ningún segmento es razonablemente alto, se realiza una segunda pasada del algoritmo, que consistirá en iniciar una segmentación incremental desde dicha colección reducida de puntos. Con este mecanismo conseguimos que, si por cualquier circunstancia, la detección de esquinas no ha sido lo suficientemente buena, es decir, no se han detectado todas las esquinas que aparecen en la escena, se pueda llegar a una segmentación 3D total de la misma.

4.5. Interfaz Gráfica

Como anteriormente describimos, este proyecto se ha dividido en dos esquemas de la arquitectura jde.c. Esta distinción hace que puedan funcionar ambos por separado, ya que son independientes entre sí. Cada esquema cuenta con su propia interfaz gráfica o GUI.

4.5.1. GUI Simulador Visual

Cuando cargamos este esquema, el GUI (figura 4.11(a)) nos ofrece en la parte superior izquierda dos imágenes, las imágenes son el resultado de la simulación, en la parte inferior de cada imagen aparecen los parámetros extrínsecos asociados a la cámara virtual que las produce. En la parte media derecha se ha dispuesto una botonera para ofrecer toda la funcionalidad del simulador y permitiendo elegir el modo de funcionamiento, cámaras en configuración estéreo o independientes, posición, punto de atención y distancia focal.



(a) Simulador

Figura 4.11: Interfaz gráfica del simulador

Para modular la escena simulada disponemos de botones para activar o desactivar

elementos como la rejilla del suelo, el punto origen del mundo o los cubos. Los cubos pueden ser colocados en la posición 3D del mundo que nos convenga, para ello se utilizan los *sliders* etiquetados como X,Y,Z.

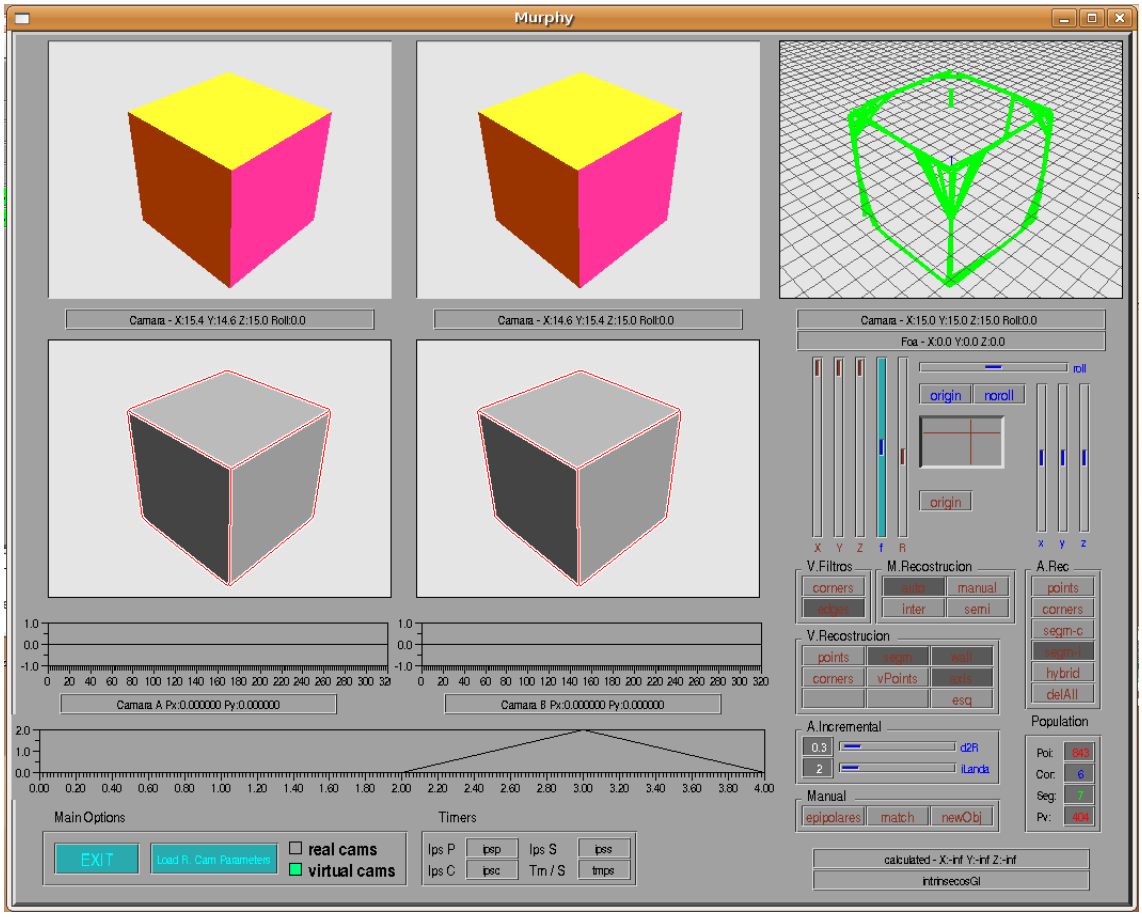
El simulador también ofrece la posibilidad de generar imágenes dinámicas, imágenes con movimiento, para activar el movimiento aparece el botón de rotación, que permite rotar los cubos de la escena sintética en sus 3 grados de libertad.

4.5.2. GUI Esquema de Reconstrucción

.

Al cargar el esquema murphy aparece el siguiente interfaz (figura 4.12(a)). En la parte superior izquierda aparecen 4 imágenes, las 2 primeras son las que corresponden con las imágenes de entrada del algoritmo, ya sean provenientes del simulador o de lal cámaras reales. Para seleccionar la fuente de origen de las imágenes aparecen unas casillas de verificación en la parte inferior de GUI. Las imágenes que aparecen inmediatamente debajo de las de entrada son los imágenes que producen como resultado el filtro de bordes y esquinas, pudiendo seleccionar qué filtro se desea visualizar. Con estas imágenes se puede identificar fácilmente los puntos de fallo de los filtros, si es que los hubiese.

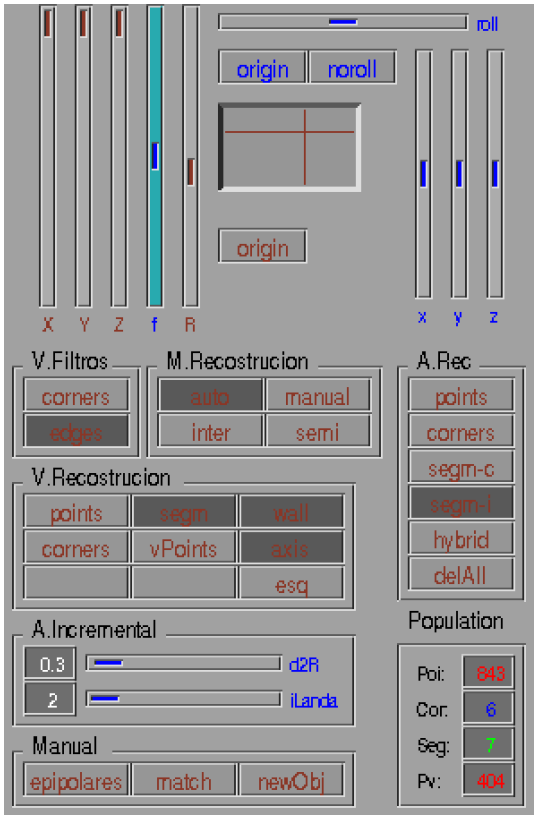
El GUI muestra en la parte superior derecha la reconstrucción generada vista desde una camara virtual móvil, permitiendo modificar los parametros de visualización elijiendo la posición y el foco de la cámara.



(a) Murphy

Figura 4.12: Interfaz del esquema de reconstrucción

El interfaz dispone de una botonera (figura 4.13(a)). En ella se puede seleccionar el algoritmo de reconstrucción a emplear, el puntual, el segmentador basado en esquinas, el incremental o el híbrido. También se ofrece la posibilidad de seleccionar los elementos a mostrar en la visualización de la reconstrucción, como por ejemplo, mostrar los puntos 3D, las esquinas detectadas, los segmentos, los puntos 3D no explicados, los puntos 3D soporte de segmentos, un suelo virtual o un eje de coordenadas. Por otra parte, se dispone de un conjunto de *sliders* destinados a seleccionar la posición del punto de vista de la reconstrucción. Todo esto muestra la riqueza de la reconstrucción ya que al ser 3D puede ser vista desde el punto que deseemos.



(a) Botonera Murphy

Figura 4.13: Botonera del esquema de reconstrucción

El interfaz nos ofrece la posibilidad de modificar los parámetros que utilizan los algoritmos de segmentación incremental e híbrido. Para ello de ha dispuesto de dos *sliders* que permiten seleccionar la distancia umbral al segmento hipotético y el tamaño de los intervalos de ocupación por unidad de medida.

Adicionalmente, a modo informativo se muestra la posición y el punto de atención de la cámara que visualiza la reconstrucción así como las poblaciones de puntos, esquinas, segmentos y puntos soporte.

Como funciones puramente académicas y de depuración se han dispuesto las siguientes funcionalidades utilizables desde el GUI:

- *Selección manual de píxeles a triangular;* Seleccionando el modo manual y picando en cada imagen en el píxel que deseemos, se calcula el punto 3D asociado a la pareja, mostrando las coordenadas del punto 3D calculado y su vista en la reconstrucción. Adicionalmente se muestran en la parte inferior de las imágenes

filtradas los gráficos de correlación asociados a cada imagen. También podemos seleccionar la aparición en las imágenes de la restricción epipolar, es decir, que muestre la epipolar sobre la que se deben encontrar los puntos seleccionados, esta funcionalidad es aplicable a todos los modos a excepción del automático.

- *Selección semiautomática de píxeles a triangular;* si seleccionamos el modo semiautomático se permite la selección manual en la imagen de la cámara A del píxel a triangular y la aplicación calcula su homólogo y lo triangula mostrando también los gráficos de correlación. Con esta funcionalidad se pueden comprobar si el algoritmo de selección de homólogos funciona o no adecuadamente.

- *Modo interactivo;* Con el modo interactivo se permite trazar el algoritmo de reconstrucción puntual, mostrando iterativamente los puntos seleccionados a triangular con sus gráficos de correlación y el punto 3D en el que desemboca permitiendo al usuario decidir si el punto 3D se añade o no a la colección de puntos. Esto permite detectar el origen de puntos 3D espúreos.

Capítulo 5
Experimentos

En este capítulo presentamos las pruebas experimentales a las que se han sometido los algoritmos, así como las resultados obtenidos. También se han caracterizado los factores que provocan pérdida de precisión en los algoritmos.

5.1. Algoritmo de reconstrucción puntual

Hemos sometido al algoritmo de reconstrucción puntual a diferentes pruebas. En la primera le pusimos como reto reconstruir un único cubo y obtuvimos como resultado la reconstrucción de la (figura 5.1).

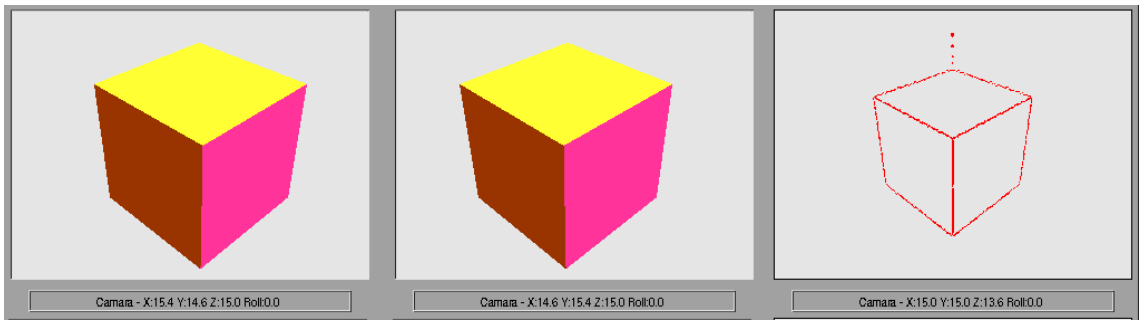


Figura 5.1: Resultados del algoritmo de reconstrucción puntual.

Como se puede comprobar, la reconstrucción desde el punto de vista de las cámaras originales resulta ser muy buena, por ello añadimos un grado más de complejidad a la escena y obtuvimos también unos resultados excelentes (figura 5.2).

Si observamos la reconstrucción desde un punto de vista favorable (posición similar a la cámaras originales) la reconstrucción parece perfecta. Sin embargo, si tomamos un punto de vista diferente se puede observar que los bordes que antes parecían líneas ahora están formados por pequeños segmentos paralelos (figura 5.3). Este efecto es

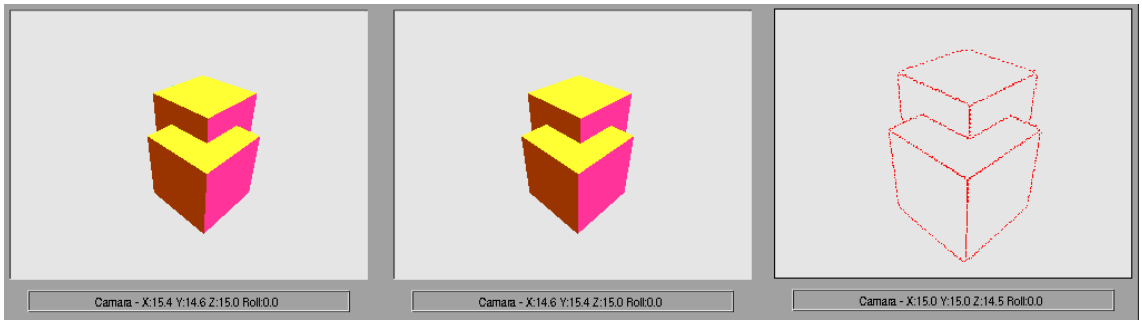


Figura 5.2: Resultados del algoritmo de reconstrucción con figura compleja.

debido a la discretización de las imágenes, lo que introduce un pequeño error en la recta de proyección asociada, derivando en el fenómeno mostrado.

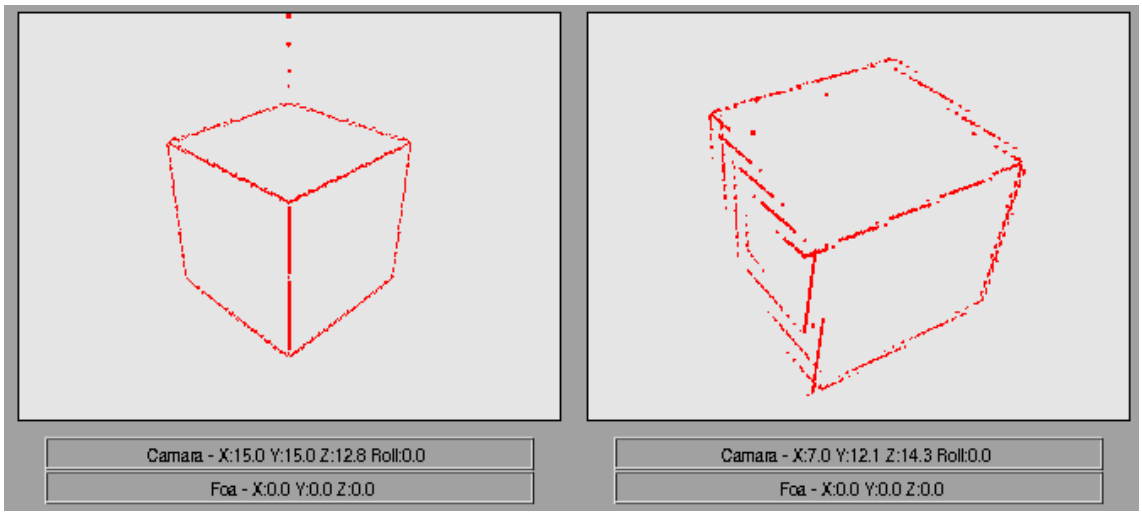


Figura 5.3: Errores introducidos por la discretizacion de las imágenes.

Se ha detectado que en ocasiones aparecen reconstrucciones incompletas de la escena (figura 5.4). Esto sucede cuando algún borde no es detectado por el filtro de bordes. Dada la naturaleza del funcionamiento del filtro, el fenómeno se produce cuando los bordes de los objetos están encerrados entre áreas de idéntica luminosidad. Una zona roja y otra zona verde de la misma luminosidad aparecerán como una única zona uniforme para el filtro de bordes, que no conseguira distinguir la frontera entre ambas. Esta deficiencia es heredada por todos los algoritmos desarrollados.

En algunas ocasiones se detectan pequeñas colecciones de puntos que no corresponden con la escena (figura 5.5). Esto sucede cuando se producen errores al determinar los pares de puntos homólogos. Los errores aparecen en entornos homogéneos en los que es difícil desambiguar entre los posibles puntos homólogos,

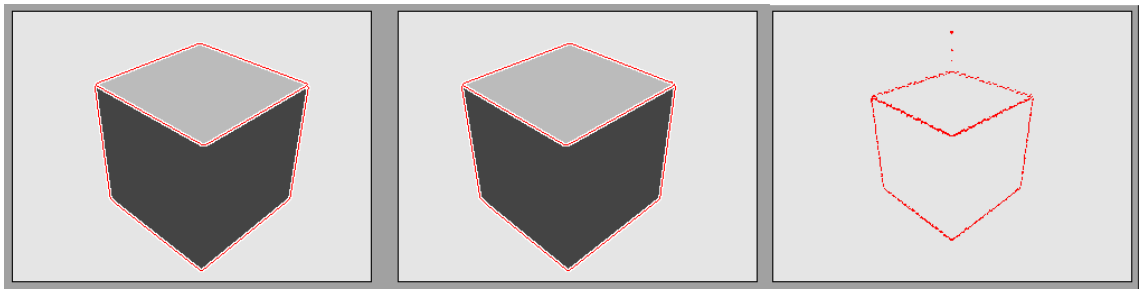


Figura 5.4: Reconstrucción incompleta a causa del filtro de bordes.

lo que provoca puntos 3D erróneos a lo largo de la línea de proyección de la cámara izquierda.

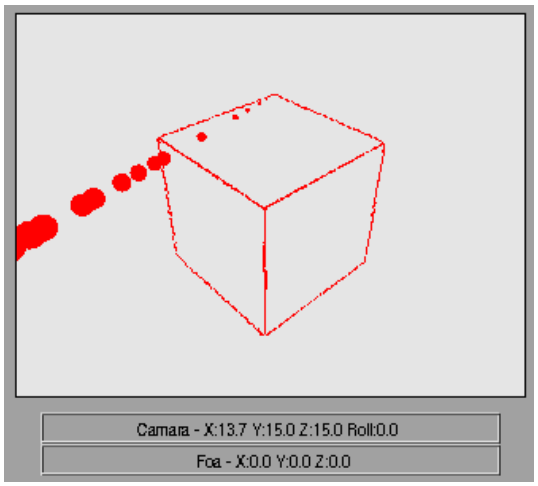


Figura 5.5: Colección de puntos espúreos por errores en la correlación.

El algoritmo también se ha probado en entornos dinámicos con un ordenador dotado de un procesador Intel Centrino a 1.6GHz, obteniendo tasas de refresco de la población de puntos 3D entre 7 y 10 veces por segundo, lo que proporciona una respuesta muy vivaz, consiguiendo reconstrucciones muy completas en las que aparacen colocados en 3D todos los bordes detectados. Con un grado de precisión del orden de 5-8 centímetros incluso con imágenes en movimiento.

5.2. Segmentación basada en esquinas

Hemos probado el comportamiento de este algoritmo realizando baterías de pruebas en diferentes escenarios. Exponemos las experiencias mas relevantes:

En escenas estáticas el algoritmo obtiene muy buenos resultados (figura 5.6), y si observamos la reconstrucción desde un punto de vista diferente al de las cámaras no se detecta ninguna anomalía (figura 5.7).

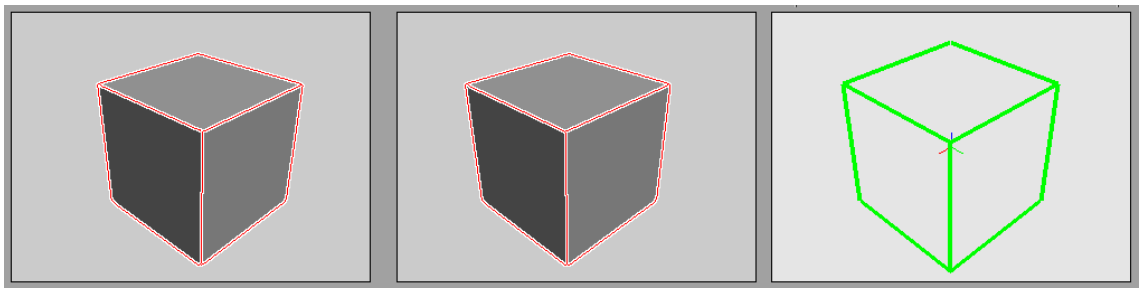


Figura 5.6: Ejecución típica del algoritmo.

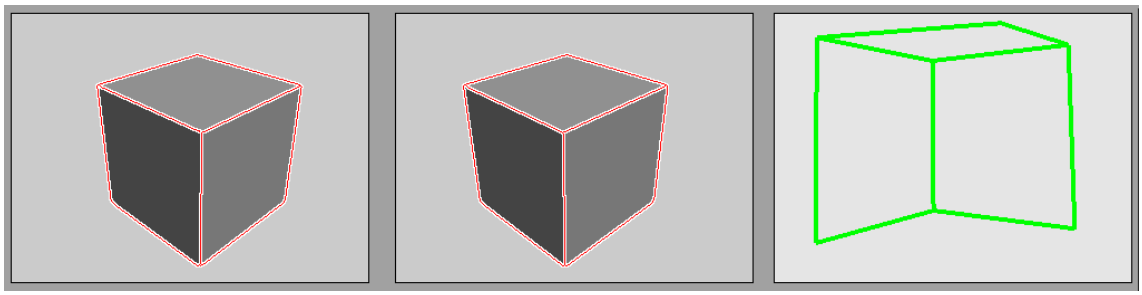


Figura 5.7: Vista posterior de la reconstrucción.

Este algoritmo utiliza como primitivas las esquinas y puntos 3D detectados que resultan de los filtros. Si alguno de estos filtros falla, no detecta todos los píxeles bajo las características descritas, el algoritmo obtiene una reconstrucción incompleta. El grado de dependencia del algoritmo con el filtro de esquinas es muy elevado. Podemos mostrar, a modo de ejemplo, un caso en el que se observan dos reconstrucciones incompletas, una a causa del filtro de esquinas (figura 5.8) y la otra por el de bordes (figura 5.9). La causada por el filtro de bordes se debe a no poder dar soporte al segmento con los

puntos del borde ya que esos puntos no han sido triangulados. Esta problemática viene heredada de la reconstrucción puntual. El algoritmo en sí no es capaz de segmentar escenas que no puedan ser dibujadas realizando trazos entre esquinas.

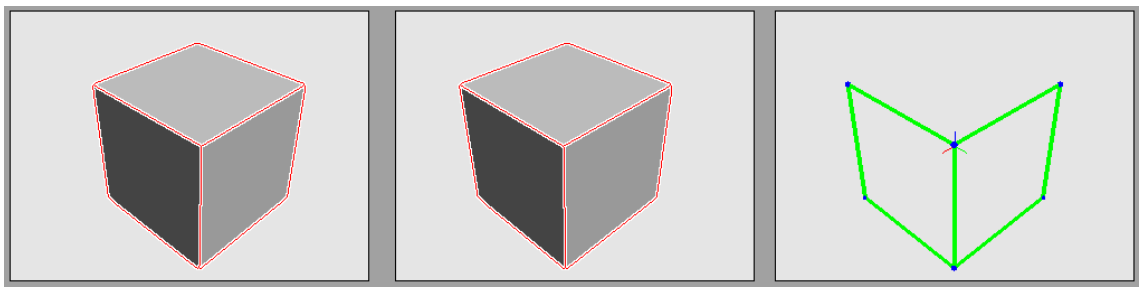


Figura 5.8: Reconstrucción incompleta por fallo en la detección de esquinas.

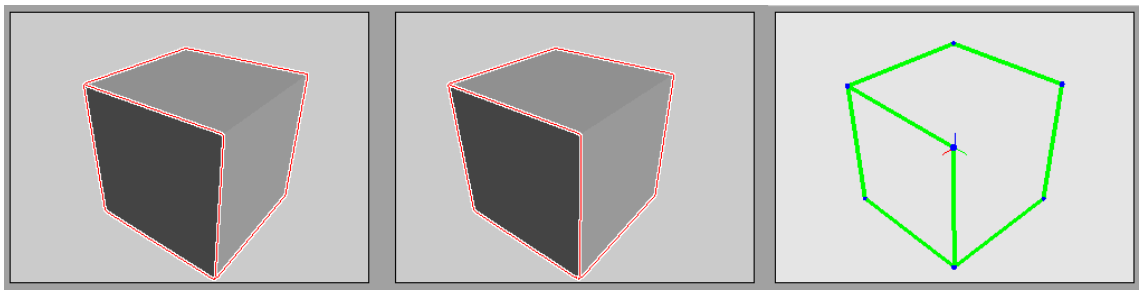


Figura 5.9: Reconstrucción incompleta por fallo del filtro de bordes.

El algoritmo ha sido probado con escenas en movimiento obteniendo un refresco de entre 6 a 8 iteraciones por segundo,lo que le hace ser muy dinámico. La fluidez que ofrece es debida a que el número de segmentos hipotetizados es pequeño.

5.3. Reconstrucción incremental

Para evaluar el comportamiento de este algoritmo, se han realizado diversas pruebas tanto estáticas como dinámicas. Rescatamos las más interesantes.

Probando el algoritmo con una escena típica, se observa que el funcionamiento es excelente (figura 5.10) aunque algo más sucio el basado en esquinas. Visualizando la escena reconstruida desde un punto diferente se puede observar que los bordes están

compuestos por diferentes segmentos no alineados (figura 5.11). Este fenómeno deriva del que se producía en la reconstrucción basada en puntos, ya que son los puntos que se segmentan.

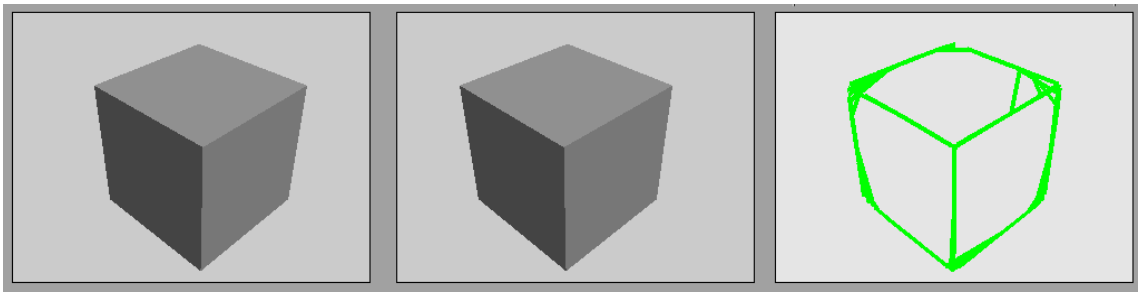


Figura 5.10: Ejecución típica de la segmentacón incremental.

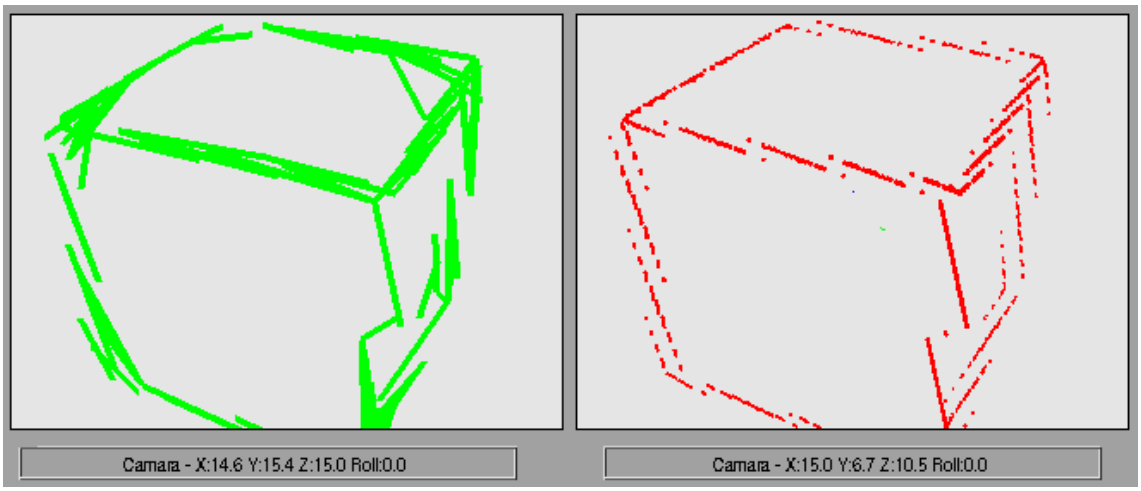


Figura 5.11: Fenómeno producido por la discretización.

En la reconstrucción se puede observar que en las cercanías de las esquinas aparecen lo que denominamos “atajos” (figura 5.12). estos atajos son segmentos 3D que unen dos segmentos que conforman una esquina. La explicación de este comportamiento es sencilla; esos segmentos piden ser validados por puntos de cada borde, lo que hace satisfacer la restricción del umbral de puntos cercanos por unidad de medida (figura 5.12). Salvo estos atajos, la segmentación es perfecta. Los segmentos no están alineados, pero porque los puntos 3D no lo están. El algoritmo segmentador los agrupa perfectamente.

Los umbrales, tanto de cercanía como en granularidad de la densidad de puntos, pueden ser ajustados a voluntad. Si estos umbrales son más exigentes, la reconstrucción

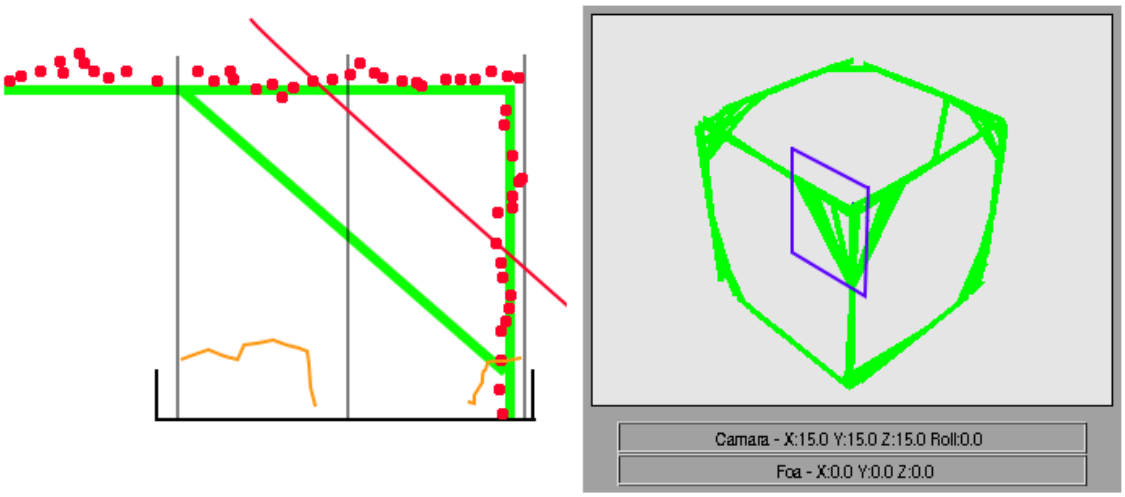


Figura 5.12: Atajos producidos en el algoritmo incremental.

será más limpia y no aparecerán atajos.No obstante, no es recomendable establecer umbrales muy exigentes ya que hacen que el tamaño del problema decrezca más lentamente, reduciendo la velocidad del algoritmo.

Optando por un compromiso entre limpieza (ausencia de atajos) y velocidad, una reconstrucción necesita del orden del medio segundo. La reconstrucción que ofrece este algoritmo en estas condiciones es completa, lo que hace que sea muy robusta. Esta implementación ofrece segmentaciones completas sin dependencia del filtro de esquinas, lo que le permite obtener segmentaciones mas completas que las del algoritmo basado en esquinas.

5.4. Segmentación híbrida

Este algoritmo combina las ideas aportadas en los dos algoritmos anteriores para sacar lo mejor de cada una.

En una ejecución típica fig observamos el excepcional comportamiento del algoritmo. Colocando la vista de la reconstrucción desde una posición desfavorable se detecta que el comportamiento en algunos bordes es errático. Este fenómeno es el mismo que se producía en la segmentación incremental. Solo aparece en los casos en

los que la primera pasada del algoritmo no ha sido capaz de segmentar completamente la escena.

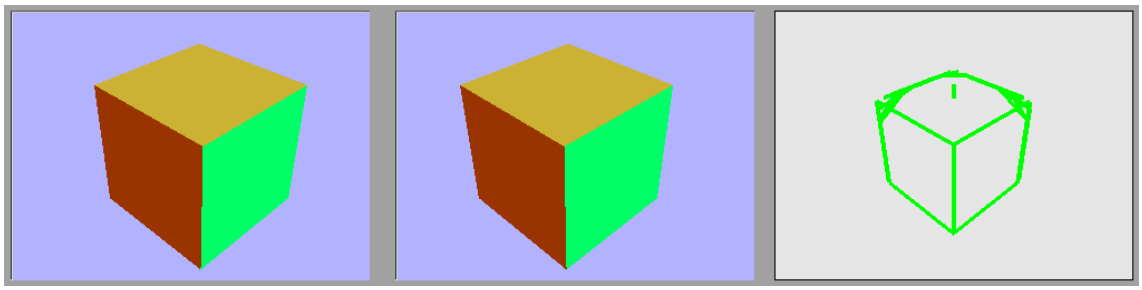


Figura 5.13: Ejecución típica de la segmentacón hibrida.

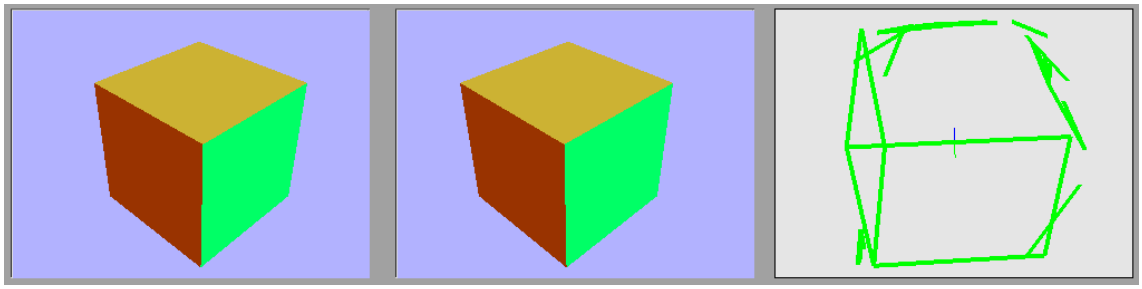


Figura 5.14: Segmentación desde punto de vista desfavorable.

Este algoritmo no es dependiente de la correcta detección de esquinas para obtener una segmentación total de la escena. El tiempo de cómputo sí se ve afectado por la detección correcta de esquinas; a mayor número de esquinas detectadas menor tiempo de computo, por lo que en una escena estándar se obtiene muy rápidamente reconstrucciones completas, ejecutándose en menos tiempo que el incremental puro. Pudiendose consegir del orden de 2 iteraciones por segundo.

Capítulo 6
Conclusiones y trabajos futuros

Descrita la solución propuesta y comentados algunos de los experimentos más relevantes, terminamos esta memoria haciendo balance de los algoritmos, exponiendo las conclusiones obtenidas en este proyecto y las posibles líneas futuras en las que se puede seguir investigando.

6.1. Conclusiones

El objetivo principal de este proyecto era tratar de realizar una *reconstrucción 3D esquemática del entorno de un robot* desde la información de un par estéreo de cámaras. Para conseguirlo se han implementado diferentes soluciones como *la reconstrucción basada en puntos* o los tres algoritmos de *segmentación* 3D.

Para llevar a cabo dicho objetivo se fueron satisfaciendo los diferentes subobjetivos:

1. *Implementación de un simulador visual de escenas 3D.* Se ha implementado un simulador visual para la plataforma jde.c que funciona como un esquema independiente, enriqueciendo la plataforma. Se ha programado utilizado OpenGl con el fin de realizar los cálculos de *renderización* a través de la tarjeta gráfica y así minimizar el consumo de CPU. Cabe destacar la labor que ha jugado el simulador. Sin haber desarrollado esta herramienta no se podrían haber conseguido los resultados obtenidos. Adicionalmente, permite trabajar con cámaras independientes a pesar de no ser objetivo del mismo.

2. *Implementación de algoritmo de reconstrucción 3D con puntos.* Se ha programado un algoritmo de reconstrucción basado en puntos que utiliza las técnicas clásicas de reconstrucción y correlación. En entornos dinámicos el algoritmo ha conseguido resultados excelentes en vivacidad.

3. *Implementación de algoritmos de reconstrucción 3D con segmentos.* Una de las partes más genuinas de este proyecto ha sido la solución dada al problema de la reconstrucción 3D basada en segmentos. Se han diseñado tres algoritmos basados en la generación de segmentos 3D hipotéticos para su posterior validación.

   a) *Segmentación basada en esquinas:* Hipotetizando solamente entre esquinas.

   b) *Segmentación incremental:* Sin hipótesis de partida, todos con todos, eliminando los ya explicados por segmentos 3D válidos.

   c) *Segmentación híbrida:* Mezcla el algoritmo basado en esquinas y el incremental.

4. *Experimentación.*
   Las implementaciones de los algoritmos de reconstrucción descritos se fueron probando iterativamente mientras se construían, describiéndose los experimentos más destacables del capitulo 5. De las experiencias recogidas hacemos un breve análisis.

A modo de balance concluimos que el algoritmo de puntos es robusto, al igual que el incremental y el híbrido. Por el contrario, el algoritmo de segmentación basado en esquinas es muy frágil; en cuanto los filtros tienen problemas no es capaz de generar una reconstrucción completa.

Otro aspecto importante es la vivacidad. Podemos observar que todos los algoritmos son vivaces a excepción del algoritmo incremental cuando aplica criterios de segmentación exigentes.

Se ha caracterizado el problema derivado de la discretizacion de las imágenes ya que este fenómeno provoca que aparezcan pequeños errores al determinar la profundidad. Este problema se pone de manifiesto cuando la reconstrucción se muestra desde un punto de vista diferente al de las cámaras originales.

La representación basada en puntos es poco compacta, con demasiados elementos para determinar la escena. Sin embargo, las basadas en segmentos son mucho mas compactas.

Concluimos que el algoritmo puntual, a pesar de ser poco compacto, es robusto y dinámico a la vez. En cuanto a los algoritmos de segmentación, destacamos como mejor

solución el híbrido, que es robusto y ofrece tiempos de respuesta buenos en la mayoría de los casos.

Haciendo un repaso tanto de los objetivos como de los requisitos que se plantearon, este proyecto ha solventado satisfactoriamente todos ellos. Se han conseguido implantaciones vivaces de todos los algoritmos propuestos ejecutándose bajo hardware convencional y ofreciendo precisión centimetrica.

Otro logro a destacar es la generalidad dada al simulador visual que hace que pueda ser utilizado como una herramienta más de la plataforma jde.c, enriqueciéndola notablemente.

6.2. Trabajos futuros

En este apartado se detallan algunas posibles mejoras que podrían realizarse sobre este proyecto y que pueden servir como nuevas líneas de investigación para futuros trabajos.

Se pueden incorporar mejoras en cuanto a las limitaciones de esta implementación, como puede ser la construcción de filtros más robustos o conseguir minimizar los errores de correlación en entornos homogéneos. Se propone también investigar con nuevas primitivas de reconstrucción y evolucionar los segmentos a triángulos para poder representar superficies.

Es inevitable plantear extrapolar la aplicación a imágenes reales. Esta labor no seria muy compleja ya que el diseño de la aplicación esta abierto al cambio dado que el diseño software esta muy cuidado. De echo a pesar de no formar parte de los objetivos de este proyecto los algoritmos de reconstrucción pueden tomar como imágenes de entrada las procedentes de las cámaras reales. Por lo que solo será necesario diseñar algoritmos destinados a la calibración de las cámaras. Se puede usar como referencia herramientas como *ARtoolkit*[1].

[1] http://www.hitl.washington.edu/artoolkit

Dado que con esta implementación somos capaces de reconocer el entorno, se podría desarrollar una aplicación capaz de permitir a los robots soportados por jde.c moverse con total libertad sobre el mundo, utilizando únicamente la información aportada por el algoritmo de reconstrucción 3D.

Bibliografía

[Barrera *et al.*, 2005] Pablo Barrera, José María Cañas, y Vicente Matellán. Visual object tracking in 3d with color based particle filter. *Int. Journal of Information Technology*, 2005.

[Barrera y Cañas, 2004] Pablo Barrera y José María Cañas. Seguimiento tridimensional usando dos cámaras. 2004.

[Cadahía, 2006] Olmo León Cadahía. Navegación de un robot con un sistema de antención visual 3d. *Proyecto Fin de Carrera, URJC*, 2006.

[de la Casa Puebla, 2005] Marta Martínez de la Casa Puebla. Sistema de atención visual en escena. *Proyecto Fin de Carrera, URJC*, 2005.

[Hartley y Zisserman, 2004] Richard Hartley y Andrew Zisserman. *Multiple View Geometry in Computer Vision.* 2004.

[Herencia, 2004] Ricardo Ortiz Herencia. Comportamiento sigue persona con visión. *Proyecto Fin de Carrera, URJC*, 2004.

[J.M López Valles, 2005] M. Fernñandez J.M López Valles, A. Fernández Caballero. Conceptos y técnicas de estereovisón por computador. *Revista Iberoamaericana de Inteligencia Artificial*, 2005.

[Martín, 2002] Félix San Martín. Comportamiento sigue pelota en un robot con visión local. *Proyecto Fin de Carrera, URJC*, 2002.

[Martínez, 2007] Iván Gracía Martínez. Reconstrucción 3d visual con algoritmos evolutivos. *Proyecto Fin de Carrera, URJC*, 2007.

[Maya, 2006] Ricardo Palacios Maya. Representación rica de la escena 3d alrededor de un robot móvil. *Proyecto Fin de Carrera, URJC*, 2006.

[Mühlmann *et al.*, 2001] Karsten Mühlmann, Dennis Maier, Jürgen Hesser, y Reinhard Männer. Calculating dense disparity maps from color stereo images, an efficient

implementation. In *IEEE Conference on Computer Vision and Pattern Recognition*, Kauai Marriott, Hawaii, December 2001.

[Nillson, 1971] N.J. Nillson. *Problem Solving Methods in Artificial Intelligence.* McGraw Hill, New York, 1971.

[P.Bustos, 2003] P.Bustos. *Murphy: hacia un robot con visión estereoscópica.* PhD thesis, Universidad de Extremadura, 2003.

[Plaza, 2003] José María Cañas Plaza. *Jerarquía Dinámica de Esquemas para la generación de comportamiento autónomo.* PhD thesis, Universidad Politécnica de Madrid, 2003.

[Plaza, 2004] José María Cañas Plaza. Manual de programación de robots con jde. *URJC*, 2004.

[S. Birchfield, 1999] C. Tomasi S. Birchfield. Depth discontinuities by pixel-to-pixel stereo. *International Journal of computer Vison*, 1999.

[Thrun *et al.*, 1999] Sebastian Thrun, M. Bennewitz, W. Burgard, A.B. Cremers, Frank Dellaert, Dieter Fox, D. Haehnel, Chuck Rosenberg, Nicholas Roy, Jamieson Schulte, y D. Schulz. Minerva: A second generation mobile tour-guide robot. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA'99)*, 1999.

[Victor M. Gómez, 2003] Vicente Matellán Victor M. Gómez, José M. Cañas. Vision based schemas for an autonomous robotic soccer player. *IV Workshop de Agentes Físicos WAF-2003*, 2003.